Modelling and Measurement in Synthetic Biology

Modelling and Measurement in Synthetic Biology

William Waites

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2020

Abstract

Synthetic biology applies engineering principles to make progress in the study of complex biological phenomena. The aim is to develop understanding through the praxis of construction and design. The computational branch of this endeavour explicitly brings the tools of abstraction and modularity to bear. This thesis pursues two distinct lines of inquiry concerning the application of computational tools in the setting of synthetic biology. One thread traces a narrative through multi-paradigm computational simulations, interpretation of results, and quantification of biological order. The other develops computational infrastructure for describing, simulating and discovering, synthetic genetic circuits.

The emergence of structure in biological organisms, morphogenesis, is critically important for understanding both normal and pathological development of tissues. Here, we focus on epithelial tissues because models of two dimensional cellular monolayers are computationally tractable. We use a vertex model that consists of a potential energy minimisation process interwoven with topological changes in the graph structure of the tissue. To make this interweaving precise, we define a language for propagators from which an unambiguous description of the simulation methodology can be constructed. The vertex model is then used to reproduce laboratory results of patterning in engineered mammalian cells. The assertion that the claim of reproduction is justified is based on a novel measure of structure on coloured graphs which we call *path entropy*. This measure is then extended to the setting of continuous regions and used to quantify the development of structure in house mouse *(Mus musculus)* embryos using three dimensional segmented anatomical models.

While it is recognised that DNA can be considered a powerful computational environment, it is far from obvious how to program with nucleic acids. Using rule-based modelling of modular biological parts, we develop a method for discovering synthetic genetic programs that meet a specification provided by the user. This method rests on the concept of annotation as applied to rule-based programs. We begin with annotating rules and proceed to generating entire rule-based programs from annotations themselves. Building on those tools we describe an evolutionary algorithm for discovering genetic circuits from specifications provided in terms of probability distributions. This strategy provides a dual benefit: using stochastic simulation captures circuit behaviour at low copy numbers as well as complex properties such as oscillations, and using standard biological parts produces results that are implementable in the laboratory.

Lay Summary

This thesis is about applying mathematics and computer science to problems in synthetic biology. Biological processes are complex; the interplay between many different processes produces the astounding variety of structures and behaviour that we see in nature. This complexity means that biological phenomena are difficult to understand and model in their entirety. Synthetic biology adopts an engineering approach to developing such an understanding. By creating and manipulating individual features of biological processes, we hope to gain insight into their properties in isolation. These insights open possibilities for the development of novel applications such as the synthesis of new kinds of biofuels or medicines. We look at two distinct lines of inquiry relating to the role of computation in this enterprise.

The first line of research begins with a proposal for some notation for describing computer simulation of certain kinds of mathematical models. We use this notation to describe a model of the dynamics of epithelial tissues modified to reproduce laboratory results of bio-engineered cells displaying pattern formation. Comparing the patterns observed in laboratory images to those produced in simulation is not straightforward. We develop a mathematical method for quantifying patterning in a principled way in order to be able to make this comparison. We then extend this method to the more complex setting of three dimensional anatomical images of house mouse *(Mus musculus)* embryos to create a quantitative narrative of the emergence of structure as it develops.

The second line of research concerns synthetic genetic circuits. Though it is possible to synthetically create DNA sequences and implant them in a host organism (usually bacteria or yeast) it is much more difficult to design these sequences. We describe a processing infrastructure to assist with this design, based on a rule-based method simulating molecular interactions. We introduce annotations on rules so that objects in simulation can be referred back to the substances in the physical world that they represent. We then define a variant of the annotation language for describing genetic circuits together with a compiler that generates the rules corresponding to the description. Finally we present an evolutionary algorithm for discovering synthetic genetic circuits that can be built from a library of standardised genetic parts.

Acknowledgements

I am truly grateful to everyone without whose support this thesis would never have been completed. Vincent Danos, my supervisor, took a chance on me, and hired me as a research associate, introducing me to a host of fascinating problems in modelling for Synthetic Biology and then accepted me as a PhD student, enabling me to study the topic in greater depth. I also thank Gordon Plotkin, my second supervisor, for his gentle yet uncompromising criticism, accepting nothing less than correct, and keeping me focused on the task at hand.

Special thanks to Peter Buneman, who first invited me to join the School of Informatics a decade ago as a visiting researcher. It was Peter who first suggested that I should pursue graduate studies and without his encouragement and mentorship, I feel that I could not have successfully undertaken this work.

Matteo Cavaliere introduced me to evolutionary game theory which proved to be such a fruitful perspective from which to look at tissue dynamics and the discovery of synthetic genetic circuits and who tutored me on the fundamentals of molecular biology. Jamie A. Davies provided suggestions and valuable insight on the questions of interest in biology and anatomy, contributing greatly to the interdisciplinary character of this thesis. Dmytri Kleiner introduced me to the concept of the artist as a worker of boundaries, which underlies my approach to interdisciplinary work.

I am grateful to the Edinburgh Rule-Based modelling group for many interesting and spirited discussions, Allister Clisham for proofreading and critical feedback, and the the Edinburgh Hacklab for an outside perspective and plenty of caffeinated beverages. The criticism of the anonymous reviewers of the portions of this thesis already published vastly improved the quality of the reported results and their justifications.

Most importantly, I thank my family, in particular my mother and my son Archer, for encouragement and patience beyond what could reasonably have been expected of anyone. And Max as well. I certainly could not have done it without them.

I also acknowledge financial support from the University of Edinburgh School of Informatics, Engineering and Physical Sciences Research Council (EPSRC) grant EP/J02175X/1, the National Academies Keck Futures Initiative of the National Academy of Sciences Award grant number NAKFI CB12, the European Union's Seventh Framework Programme for research, technological development and demonstration grant number 320823, and the UK Research Councils' Synthetic Biology for Growth programme, the BBSRC, EPSRC and the MRC.

Declaration

I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contributions and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

The work presented in Chapter 3 is reprinted with permission from William Waites, Matteo Cavaliere, Élise Cachat, Vincent Danos and Jamie A. Davies, "An information-theoretic measure for patterning in epithelial tissues", *IEEE Access* (2018)[1]. The work was conceived by all of the authors. I programmed and conducted the simulations that were able to reproduce the biological results, originated and formulated the concept of *Path Entropy* for comparing the simulated and laboratory data, implemented it in software, conducted the numerical experiments, produced the figures apart from those originating with the laboratory experiments, and drafted the main text.

The work presented in Chapter 4 is reprinted with permission from William Waites and Jamie A. Davies, "Emergence of Structure in Mouse Embryos: Structural Entropy Morphometry Applied to Segmented Anatomical Models", *Journal of Anatomy* (2019)[2]. The work was conceived by both authors. I developed the mathematical formulation of *Structural Entropy* by extending *Path Entropy* to the continuous setting, implemented the software for processing the segmented anatomical data, conducted the analysis and drafted the initial text; both authors contributed equally to the final text.

The work presented in Chapter 5 is adapted by permission from Springer Nature: *Modelling Biomolecular Site Dynamics* of *Methods in Molecular Biology* (ed. William Hlavacek), "Chapter 13: Annotations for Rule-Based Models" by Matteo Cavaliere, Vincent Danos, Ricardo Honorato-Zimmer William Waites (2019)[3]. The work was conceived by all of the authors and I drafted much of the chapter text. In particular, I contributed the discussion of reactions and rules, the relationship of annotations to the objects that are being annotated, and the relationship of the concepts of abstraction and annotation. This work was derived from a previous paper published in the journal *Bioinformatics* as "Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization" by Göksel Mısırlı, Matteo Cavaliere, William

[1] https://ieeexplore.ieee.org/abstract/document/8405520
[2] https://onlinelibrary.wiley.com/doi/full/10.1111/joa.13031
[3] https://link.springer.com/protocol/10.1007/978-1-4939-9102-0_13

Waites, Matthew Pocock, Curtis Madsen, Owen Gilfellon, Ricardo Honorato-Zimmer, Paolo Zuliani, Vincent Danos and Anil Wipat (2015)[4]. I became involved with that paper at the beginning of my PhD studies after it had been returned from *Bioinformatics* for major revisions. My contributions were the method of adding annotations in concrete rule-based languages in a backwards compatible way, implementation of the `krdf` software for extracting and processing these annotations, writing queries to demonstrate how useful information can be extracted from the annotations, generation of the contact map diagram and significant portions of the final text.

The work presented in Chapter 6 was previously published in the journal *ACS Synthetic Biology* as "A Genetic Circuit Compiler: Generating Combinatorial Genetic Circuits with Web Semantics and Inference" by William Waites, Göksel Mısırlı, Matteo Cavaliere, Vincent Danos and Anil Wipat (2018)[5]. The work was conceived by all of the authors. I originated the concept of using inference to materialise implicit information to achieve the goal of succinctness, designed, implemented and tested the `kcomp` compiler software and templates, produced the figures (except where otherwise noted) and drafted the majority of the text.

The work presented in Chapter 7 is joint work with Matteo Cavaliere and Vincent Danos. The work was conceived by all of the authors. I originated the concept of employing a grammar to guide the evolutionary search for synthetic genetic circuits that match a given specification, developed the mathematical formulation, implemented the `kgen` software performs the algorithm and drafted the text.

All material has either (a) not been published elsewhere; (b) has been published under an Open Access copyright license that permits re-publication with attribution, which has been duly given (Chapters 4 and 6); or (c) has been published and a specific license has been obtained (Chapters 3, and 5, License Number 4591911004613). As such all copyright requirements for publication of this PhD thesis have been met. The chapters consisting of published material have been slightly modified for stylistic reasons, such as replacing the phrase "this paper" with "this chapter", but are otherwise identical except where explicitly noted.

[4] https://academic.oup.com/bioinformatics/article/32/6/908/1743798
[5] https://pubs.acs.org/doi/abs/10.1021/acssynbio.8b00201

To Archer

Table of Contents

Chapter 1

Introduction

In his book *Trickster Makes This World* (Hyde, 2008), Lewis Hyde describes the artist as a worker of boundaries, sometimes transgressing them, or pushing them into the realm of the absurd. This characteristic of art drives the development of culture, he claims, and this thread has been present since ancient times and can be seen across cultures in the archetype of the "fool" and in trickster figures such as Nanaboozhoo, Coyote, Raven, Br'er Rabbit, Reynard, Eshu, Loki, Dionysus in various mythologies. The locus of scientific activity is also at the boundaries. When phenomena are unchanging or thoroughly understood, they are no longer very interesting. The fruitful areas of research are at the edge of what we know, where one regime of behaviour gives way to another, where entities which may be well understood on their own interact in difficult to understand ways. The boundaries of different fields of science, which are often characterised by differences in time or spatial scale, are particularly productive areas of research. Historically, these interdisciplinary boundaries are infrequently crossed, though this situation seems to be improving. The work which follows straddles this kind of boundary, with mathematics and computer science on one side and biology and anatomy on the other.

Living creatures are bewilderingly complex. Diverse processes happen within them simultaneously on many different physical and time scales. Proteins, of which there are many thousands of kinds, are produced and interact at time-scales on the order of seconds or less and their size is measured in tens of nanometers (Milo et al., 2009). Several hundred different kinds of cell undergo mitosis replicate at time scales of minutes or hours and their diameter is typically measured in the hundreds of nanometers in humans (Milo et al., 2009). These same cells arrange themselves over a period ranging from hours to months (O'Rahilly et al., 1975) and years to form a variety of

physical structures ranging from much less than a millimetre to more than a meter in extent. These processes do not operate in isolation, they interact with each other. Proteins produced by genetic machinery traverse cell membranes and interact with the external universe, perhaps stimulating or suppressing the division of other cells and growth of tissues. Even without considering cognitive and psychological phenomena whose relationship to the physical substrate remains unclear, the complete set of such interactions is vast.

To make progress in understanding the dynamics within biological organisms a dual approach is necessary. The experimental approach is to isolate one process or behaviour in the laboratory as far as possible and making observations and measurements varying external influences in a carefully controlled way. This yields a wealth of data, but can have little explanatory power; it tells us about what happens, but does little to elucidate the mechanism. A good example of this is the commonly available drug paracetamol (acetaminophen). First synthesised 140 years ago, paracetamol has a relatively simple chemical structure and has been the subject of innumerable scientific studies, yet its mechanism of action, the nature of the causal relationship between ingesting the drug and its effects, is not well understood. There is consensus that it has an anti-inflammatory effect (Botting et al., 2005; Aronoff et al., 2006) through suppression of enzymes called cyclooxygenases that promote an immune response, but that effect varies throughout the body for unknown reasons (Ghanem et al., 2016). Nevertheless experimental data is immensely valuable. For clinical and for public health education purposes it is very useful to know for sure that paracetamol relieves certain kinds of pain and that overuse leads to liver disease (Larson et al., 2005). But that knowledge is not very predictive of scenarios that are even slightly different; it does little to illuminate any general principles.

To provide explanatory power, the theoretical approach uses models and abstraction to distil the salient features of the mechanisms underlying the experimental measurements. For a model to be useful for explanation it should be as simple as possible but not so simple that it omits essential features. In the same way as with experiment, where aspects of the dynamics of a biological system are isolated and controlled, models typically address a small subset of the behaviour. Those elements of the system that are considered external to the model or which have a sufficiently small effect on the outcome are held fixed or abstracted away. Those same external elements may, however, be the subject of different models. This insight has led to heterogeneous agglomerative models consisting of many smaller models that communicate. An example of

this is Karr's Whole Cell Model (Karr et al., 2012) which consists of 28 sub-models of processes that occur in the *M. genitalium* bacteria, each of different character, and which successfully makes predictions about a cell's phenotype from its genotype. This constructive approach to explaining complex phenomena is compelling because, rather than positing a single monolithic model, it re-uses and combines small and individually well-understood models.

In Chapter 2 we build on established mathematics of flows in dynamical systems (Katok et al., 1995) to develop a notation for describing heterogeneous computational models. This work is motivated by Karr's Whole Cell Model (Karr et al., 2012) and by Farhadifar's vertex model for epithelial tissues (Farhadifar et al., 2007). The main weakness of Karr's model is that the mechanism of combining the sub-models (which he calls "modules") is ad-hoc and unexamined. Even if the modules are individually well-defined and sound, it is necessary to understand the properties of the operation by which they are combined. This is particularly true where the modules have fundamentally different characters. Karr's model contains modules that are expressed in terms of systems of ordinary differential equations, optimisation processes, stochastic processes and procedural programs (Karr et al., 2012). It is not feasible at the present time to give a complete account of the implications of Karr's approach on the accuracy of his combined model, but we can make some progress. Our notation is designed to make the trade-off of the modular approach clear. What is gained by small sub-models in clarity and re-usability is lost in accuracy. It is possible, however, to give bounds for this composition error in some simple cases.

Despite the lack of progress on accuracy estimation for the general case of composed models of arbitrary kinds, we can show that it is possible to describe them in a clear and unambiguous way. We show how our notation can be used to express various simple kinds of numerical modelling techniques. We then apply the notation to Farhadifar's vertex model for epithelial tissues (Farhadifar et al., 2007) which is used in the following chapter. Farhadifar's model is interesting because it describes the dynamics of cellular monolayers in simple terms using two very different techniques. In this model, a tissue is represented as a graph enhanced with spatial coordinates for each vertex. A chord in the graph is a cell. The force on each vertex depends on a scalar potential energy function of the cell's area, perimeter, and the length of each edge shared with adjacent cells. For a static tissue, obtaining an equilibrium configuration is straightforward: one simply finds a local minimum of the potential. That can be achieved using any number of optimisation techniques. The model also includes topological changes. Cells divide and

reproduce, they may be extruded or removed from the tissue and under certain conditions they migrate. These operations change the space in which the potential is defined. A simulation proceeds by interleaving the optimisation process with these topological changes. To the extent that it can be difficult to tell the precise way this interleaving is done, and compare different software realisations of the same mathematical model, we can use our notation to clarify.

In Chapter 3, first published in the journal *IEEE Access* (Waites; Cavaliere, et al., 2018), we showcase the interplay between theory and experiment described above. We use a computational model to reproduce the kind of patterning observed in engineered cells in the laboratory and invent a novel information-theoretic measure defined on coloured graphs to justify the claim that the computational results correspond to the laboratory results. We begin with experimental results in a tightly packed cellular monolayer (though not exactly an epithelial tissue) containing two varieties of cells (Cachat et al., 2016). These cells are engineered such that the presence of tetracycline modifies the adhesivity on boundaries between heterotypic cells. The effect of these manipulations is the spontaneous appearance of patterning reminiscent of Turing's "dappling" (Turing, 1952) but, evidently, due only to local interactions and not long-range chemical signalling. Cachat et al.'s study evinces a certain kind of explanation which can be summarised as: the introduction of tetracycline causes more cadherin proteins to be expressed which in turn increase adhesiveness between heterotypic cells and this adhesiveness causes patterning. There is a gap in the explanation. It is not obvious how adhesiveness should cause patterning. A version of Farhadifar's model (Farhadifar et al., 2007), modified to have two kinds of cells, suggests that what is called edge tension in Farhadifar's model plays essentially the same role adhesion. Conducting simulations of this model produces results that appear visually very similar to the imagery from the experimental study.

To claim that this simple physical model explains how adhesiveness causes patterning, we need a way to quantify visual similarity. It seems "obvious" that the imagery produced by the numerical simulation and the imagery taken with a confocal microscope of the experiment are similar, but how can that be substantiated objectively? We answer this by considering the cellular monolayer as a coloured graph where the vertices are the cells themselves and the edges are their adjacencies (this graph is dual to the graph in Farhadifar's model). Using the colour sequences of paths in this graph we can define a probability distribution. Not only do we look at how likely a cell is to be, say, red or green, we also consider how likely a red cell is to be next to another red

cell or a green cell, how likely sequences of three red cells in a row are to be found among all sequences of three cells, and so forth. From these probability distributions we can calculate an information-theoretic measure of order in the graph that we call *Path Entropy*. The contention is that order and pattern are related and "more" pattern corresponds to a less uniform distribution of paths. If the first contribution of Chapter 3 is a physical explanation of patterning due to adhesion, the second, more important one is a principled way to quantify a certain kind of order that can be present in coloured graphs.

In Chapter 4, to appear in the *Journal of Anatomy* (Waites; Jamie A Davies, 2019), we we show how to extend path entropy to the setting of continuous regions and apply it to the study of developmental anatomy. The emergence of order in developing organisms is a known phenomenon (Kauffman, 1993) but what that means, precisely, has not been articulated in a systematic way (Grizzi et al., 2005). Our starting point is the analysis of the three dimensional segmented imagery from the *Edinburgh Mouse Atlas* (Richardson; Venkataraman; Stevenson; Yang; Nicholas Burton, et al., 2009) done by Jamie A Davies (2016). Davies considered the change in time of the number of distinct tissue types as the mouse embryo develops and found an exponential increase and that this count of tissue types corresponds to a certain notion of anatomical complexity. This analysis doesn't account for spatial relationships nor does it explain the qualitative increase in anatomical order. Though the tissues present in a mouse embryo are clearly made of cells, apart from perhaps at the very beginning of development, that would be too granular a level to consider in an analysis of macroscopic anatomical structure. We develop *Structural Entropy* by extending Path Entropy to this setting using the random walk of a notional particle that is free to travel through the body of the embryo. The chance that this particle is initially found in a particular tissue region or organ is proportional to the volume of that region. The chance that this particle diffuses across boundaries between tissues is proportional to the fraction of the source region's surface area that touches the destination region. We analyse the starting probability distribution and the steady state distribution for the Markov process defined by the inter-tissue transition probabilities. The entropy of these distributions, when normalised to account for the number of segments (which increases as Davies found), decreases with development. This finding appears robust despite some defects in the underlying data and furnishes an objective way to quantify the increase in order attendant with anatomical development.

In Chapter 5, first published as a chapter (Cavaliere et al., 2019) in the book *Modelling Biomolecular Site Dynamics* (ed. Hlavacek), we turn to the topic of genetic

circuits with an extended discussion on annotating rule-based models. Our focus is on rule-based models (Danos; Jérôme Feret; W. Fontana; Russell Harmer, et al., 2007; Danos; Jérôme Feret, et al., 2008) of molecular biology not only because stochastic simulation allows for the study of systems characterised by noise and low copy numbers of molecules but because of expressiveness, for example, in modelling polymer production. Chapter 5 is based on our original paper (Mısırlı; Cavaliere, et al., 2015) where we proposed a simple technique for annotating rule-based models with metadata to facilitate their re-use for the reasons just mentioned. There is an important insight in this chapter only briefly mentioned in the original paper, namely that the distinction between what is an annotation and what is an object being annotated is not so clear-cut as might at first appear. In particular, for some purposes, such as when providing a rule to a simulation program, it is convenient to consider the rule to be an object onto which annotations are placed. For another purpose, such as analysing interactions between rules, it is more convenient to consider a rule to itself be an annotation on its input and output objects, documenting their relationships. This freedom to alter perspective (Buneman et al., 2013) is quite powerful and leads to the idea that it is possible, with suitable auxiliary information, to produce rules from annotations themselves.

In Chapter 6, first published in the journal *ACS Synthetic Biology* (Waites; Mısırlı, et al., 2018), we use this relationship between annotations and rules to show to model synthetic genetic circuits in a modular way, and describe compiler infrastructure to generate rule-based models from annotated genetic circuits. Rather than rules as such, we mainly consider genetic circuits in terms of higher-level constructs: sequences of biological parts. A biological part is a functional genetic unit larger than a base pair but smaller than an operon (Canton et al., 2008; Del Vecchio et al., 2008; Shetty et al., 2008; Pedersen et al., 2009). Examples of these kinds of parts are operators, promoters, and coding sequences. Standard versions of these parts have been proposed (Cooling et al., 2010; Galdzicki; C. Rodriguez, et al., 2011) and are available from biomedical suppliers for assembly into synthetic circuits in the laboratory. Encapsulating the genetic functionality into biological parts gives us modularity. The principle of modularity underlies the compositionality described in Chapter 2 and it also means that, so long as the input-output contract is respected, a module may be freely replaced with a different module to obtain a different composite model. A key advantage of modularity in computational models is that their input-output specification or contract is well-defined. Genetic circuits are themselves computational environments (Buchler et al., 2003) and we contend that they can be more easily programmed using a modular approach.

Given a resource for biological modelling that made available a vast library of small executable models which are designed to be combined in this way, new numerical experiments could be constructed by assembling modules from this library. New modules representing specific models could be developed and tested in an otherwise well-studied environment made of standard, published modules. Such databases of biological models—the subject-matter mainly consisting of genetic parts and protein interactions—have indeed been proposed (Snoep et al., 2003; Moraru et al., 2008; Cooling et al., 2010; Li et al., 2010; T. Yu et al., 2011; Mısırlı; J. Hallinan, et al., 2014). To be most effective, to permit use of models to identify proteins, DNA sequences and so forth from different databases, the machinery of *linked data* is helpful (Heath et al., 2011; Krause et al., 2010; Klement et al., 2014). Chapter 6 contributes an annotation language for describing these assemblies and a compiler to translate the descriptions into executable simulation code. The implementation of the compiler is described in detail and makes extensive use of inference in order that the description language can remain succinct, clear, and largely free of implementation details. The result of this approach is that the compiler can produce output in different languages for simulation such as Kappa (Boutillier et al., 2018; Krivine et al., 2018) and BioNetGen (Michael L Blinov; James R Faeder, et al., 2004; James R. Faeder et al., 2009). The input can be readily transformed into a format that can be consumed by the automated laboratory assembly, or indeed read and interpreted directly. This is an essential piece of infrastructure as it enables a degree of parity between the numerical simulation and the laboratory experiment environment. Laboratory experiments can be slightly modified and studied *in silico* and numerical simulations can be reproduced *in vitro* and the results compared.

In Chapter 7, which is joint work with Matteo Cavaliere and Vincent Danos, we present the third and final piece in the processing pipeline for synthetic genetic circuits: a method for discovery genetic circuits that meet a particular specification. In the previous chapters we developed a way of annotating rule-based models and a way of generating rule-based models from annotations representing genetic circuits, but the question of how to discover which genetic circuits to describe remains unaddressed. The space of possible designs is very large and most synthetic genetic circuits are designed by hand, with a good dose of informed intuition about what ought to work. We propose an evolutionary algorithm for discovery of candidate circuits and demonstrate that it works as specified, at least in simple circumstances. Given a description of desired behaviour, a library of parts, a starting circuit and a grammar (which can be implicit in the syntax of the starting circuit), the evolutionary algorithm searches for sequences of parts that

conform best to the specification. The grammar is important because it guides the search. At each step of the algorithm, the circuit is mutated. The circuit may only mutate in a way permitted by the grammar. This eliminates large swathes of the space of possible synthetic genomes that can be determined to be *a priori* non-viable: either because they could not be constructed or because they simply would not work. The specification of the desired behaviour is also notable. We take inspiration from test-driven development here and the behaviour is specified as a series of test cases or $(input, output)$ pairs together with a method of calculating the output from the time-series produced by a simulation. The inputs are encoded as starting copy numbers of molecules participating in the rules, and the outputs, in general, as time-series. The benefit of considering entire time-series is that we can specify bi-modal equilibrium distributions and time-varying behaviour. The latter is difficult to measure in the laboratory and we demonstrate it here in simple circumstances. Using this evolutionary algorithm, and the circuit description language from the previous chapter, it is possible to systematically search for genetic circuits that implement a given behaviour which can then be synthesised in the laboratory and tested.

Finally, Chapter 8 concludes the thesis. We provide a broad perspective on the role of measurement as applied to modelling, and detail the specific contributions of the thesis. We then discuss possibilities for future research and reflect on the interdisciplinary character of our work.

Chapter 2

The φ Propagator

2.1 Representing Numerical Simulation Algorithms

In the chapters that follow, models for a variety of biological and other systems will be presented. Some of these models will be heterogeneous in the sense of being composed of several fundamentally different processes. In Chapter 3 we examine tightly packed networks of cells. Starting with the model of Farhadifar et al. (2007) we describe the continuous action of mechanical forces with differential equations via a scalar potential, and also subject the system to topological changes as the cells rearrange.

In most cases there is no closed-form solution for the time-evolution of these kinds of models. It is rare to be able to obtain any analytic solution at all and we must work mainly with difference or differential equations, or similar forms and conduct numerical simulations to see how the system behaves. For a given mathematical model there are choices about how to simulate it: for continuous sub-model, which integration scheme is best? How are instantaneous changes interspersed with phases of continuous evolution? Where there are both continuous- and discrete-time sub-models, how are these to be combined? To what extent is it possible to compare different simulations of the same model?

In order to address these questions, we need a consistent way to represent a simulation algorithm for a model. Often a simulation algorithm is simply described in words in a research article. This is unsatisfactory for a detailed understanding of the method because it is insufficiently precise. This becomes clear when implementing a model from such a description, perhaps in order to reproduce the results. If the model is even moderately complex, one is immediately faced with implementation choices on how to structure the software, which libraries for common mathematical operations to use, and

so forth. One might compare the results of such a simulation to the ones in the original article, but this does not guarantee that the simulation is the same. It can even be argued that results reproduced by a *different* technique are more robust, but it is not clear how to do this without a consistent way of talking about the similarities and differences between the techniques.

One can examine the software that implements the simulation. The software is not always available, though we wish it would be (Barnes, 2010; Peng, 2011; Ince et al., 2012). When it is available, and is of good quality, there remain barriers to using it as a tool to understand the simulation. Clarity of exposition may have been sacrificed in order to optimise for speed of execution or economy of storage. Simulations may be written in different styles in different programming languages with little possibility of any universally acceptable notion of equivalence (Blass et al., 2009; Yanofsky, 2010).

The approach that we take here takes inspiration from the mathematics of flows in dynamical systems. This field has a long distinguished history with origins in the study of the time evolution of systems operating under Newton's laws and has well established treatments of various kinds of systems in both discrete and continuous settings (Katok et al., 1995). As the goal is to facilitate concise descripions of simulation algorithms and not to extend the fundamental mathematics as such, we define a notation for *propagators*, denoted φ, to describe these algorithms. As with flows, these propagators have a semigroup structure and are endomorpisms (Chicone et al., 1999; LaSalle, 1976) which enables them to be composed, or chained together. These descriptions are implementation independent. Two propagators are considered equivalent if they implement the same model, conceived of as a particular kind of computable function. Difficulties arise, however, when determining if two propagators are indeed in the same equivalence class. We expect that numerical simulation algorithms only approximate a given function or model. Different algorithms will produce numerical different results, yet still be sufficiently similar that they can be reasonably considered to implement the same model. This can be straightforwardly resolved by thinking about equivalence as being to within a specified accuracy or tolerance, as is standard with numerical methods.

The initial motivation for the φ notation was as an *aide-mémoire*. When working with several implementations of Farhadifar's Vertex model, we frequently wondered about the fine differences between them. When returning to these software codes after a time spent on other work, it was often necessary to re-examine them closely to recall the detail of exactly how the topological transitions were interleaved with optimisation of the potential function. We found a need to be able to succinctly, yet accurately transcribe

the simulation algorithm. The notation that we define here accomplishes this. It also has the desirable property that it can be directly implemented in its own right and serve implementation in its own right.

We define our notation by extending the notation for flows that is often used in numerical integrators for differential equations (McLachlan, 1995; Hairer et al., 1996; Hochbruck et al., 1998), particularly in physics (Forest, 2006; Blanes et al., 2006), with features to describe hybrid automata or models with a mixed continuous-discrete character together with stochastic simulations. The extra features are: generalised state, or configuration space, probabilistic permutation and iteration. Generalised state is considered in work on dynamical systems (Katok et al., 1995) though systems whose time evolution rule changes (perhaps probabilistically) is a less thoroughly explored topic. Together these features enable us to capture topological changes for graph-based models as well as gradient descent and annealing methods of optimisation. We begin with some preliminaries, and proceed by describing the basic features of the noation and then demonstrate how some well-known algorithms, as well as some lesser-known ones, can be represented.

2.2 Preliminaries

2.2.1 The Model and its State

The models that we consider here are descriptions of how the state of some system changes. The configuration of the system is described by a point in a *state space*, $\mathbb{X}$, also conventionally called *configuration space*, or *phase space*. Every point state space $x \in \mathbb{X}$, contains all the information necessary to describe a possible configuration of the system. The state space contains all possible configurations. The structure of $\mathbb{X}$ can be arbitrarily complex: it can simply be a vector space of real numbers, it can be a space of sets or graphs, or functions. It can also be the disjoint union of any of these. The state space can essentially be any well-defined construct.

Deterministic models describe the time evolution of the system from some initial point in state space. Given such initial conditions, $\mathbf{s}(0) = \mathbf{x}_0$, we can obtain a unique, ordered sequence of states, $\mathbf{s}(t)$, which we call its trajectory. The parameter, t, representing time, is drawn from a commutative monoid[1], $(\mathbb{T}, 0, +, \leq)$ (Fuchs, 2011) such as the

[1] A semigroup is a set, S, with an associative binary operation, $\cdot$, such that for any $a, b, c \in S$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. A monoid is a semigroup with an identity element, 0 such that for any $a \in S$, $a \cdot 0 = a$. A commutative monoid with a binary relation, $\leq$, such that for any $a, c \in S$, if $a \leq c$ then there exists a

integers (for discrete time simulations) or the real numbers (for continuous time). We restrict the order relation to be linear so that it is possible to tell, for any $t_1, t_2 \in \mathbb{T}$, which is earlier and which is later ($t_1 \leq t_2$ or $t_2 \leq t_1$). The commutative structure guarantees that for any time $t \in \mathbb{T}$ and any time-step $h \in \mathbb{T}$ it must hold that $t + h \in \mathbb{T}$. These properties of commutative monoids are sufficient to be able to describe the system as stepping forward in time.

We formulate a deterministic model as a function with type, $\mathrm{M}_D : \mathbb{T} \to \mathbb{X} \to \mathbb{X}$. Given a time interval and a point in state space, the function produces a new point that predicts the configuration of the system after that time has passed,

$$\mathbf{x}' = \mathrm{M}_D\,(h, \mathbf{x}) \tag{2.1}$$

Repeatedly evaluating this function samples a trajectory, with a step-size of h,

$$\begin{aligned} \mathbf{s}(0) &= \mathbf{x}_0 \\ \mathbf{s}(t+h) &= \mathrm{M}_D\,(h, \mathbf{s}(t)) \end{aligned} \tag{2.2}$$

Example: Exponential Propagators

The concept of propagator arises naturally with initial value problems of the form,

$$\frac{d\mathbf{s}(t)}{dt} = A\mathbf{s}(t) \qquad\qquad \mathbf{s}(0) = \mathbf{x}_0 \tag{2.3}$$

where, in introductory texts, $\mathbf{s}$ is typically a vector valued function, $\mathbf{s} : \mathbb{R} \to \mathbb{R}^n$, and A is a matrix in $\mathbb{R}^{n\times n}$ (Arnold, 1992), though more sophisticated settings that have the same formal solution are possible (Baez et al., 2012; Behr; Danos, et al., 2016). The solution is, of course,

$$\mathbf{s}(t) = e^{tA}\mathbf{x}_0 \tag{2.4}$$

Where this class of model can be used, it is advantageous because it admits exact solutions. Points on the trajectory at any time in the solution can be directly calculated so long as the matrix exponential can be evaluated. That is not without challenges but there are known methods that give good results efficiently (Moler et al., 2003).

Suppose that we want to sample the trajectory of this system to find its configuration at equally spaced time intervals, $\mathbf{s}(t)$ for $t \in (h, 2h, 3h, \ldots)$. We can do this by writing the propagator,

$$\varphi_h = e^{hA} \tag{2.5}$$

$b \in s$ such that $a \cdot b = c$. For commutative monoids we write the binary operation as $+$ rather than as $\cdot$.

which gives,

$$\mathbf{s}(h) = \varphi_h(\mathbf{x}_0) \tag{2.6}$$

Informally, this is read as the propagator φ_h moves the system forward in time by an interval h. Repeated application of the propagator gives,

$$\mathbf{s}(nh) = \varphi_h^n(\mathbf{x}_0) \tag{2.7}$$

generating the desired trajectory.

2.2.2 Probabilistic Models and Repeatability

Not all models are deterministic. A model may be *probabilistic*: given the same initial conditions and a time interval, evaluating the model more than once can produce different results. Rather than a function whose codomain is the model's state space, a probabilistic model is a function to distributions over state space, $\mathrm{M}_P : \mathbb{T} \to \mathbb{X} \to \mathscr{D}\mathbb{X}$.

Trajectories for probabilistic models are produced by iterative evaluation just as with deterministic models. Because each evaluation is probabilistic, each trajectory is likely to be unique. For many models, the set of trajectories produced from identical initial conditions exhibit meaningful statistical properties, for example correlation or ergodicity.

In practice, probabilistic models are implemented using pseudo-random number generators. Good pseudo-random number generators produce sequences of numbers that have statistical properties identical to truly random sequences except that the sequences are repeatable when given the same starting conditions, called the random seed (Teukolsky et al., 2007). Repeatability is important for software debugging and for understanding stochastic processes (Ripley, 1990). More generally, repeatability is important for science, including numerical experiments.

The use of pseudo-random number generators yields a recipe for turning what would otherwise be a probabilistic model into a deterministic one: simply augment the model's state space with the internal state of the random number generator. Alternatively, we consider the pseudo-random number generator itself to be part of the model. This allows putatively probabilistic models to be of the same type as deterministic ones.

Though this approach is sound, there is a certain inelegance in modifying the model to include the random number generator. It seems like an unwarranted mixing of concerns, a violation of modularity, mixing a mathematical model of some physical or other process of interest, with implementation details of random number generators.

From a "rough and ready" engineering perspective this is not a problem. From a theoretical perspective it is unsatisfying.

There is a theoretical framework that solves this in a satisfying way. Moggi (1989) and subsequently Wadler (1990) show how to use monads in purely functional languages to keep state or receive external input (see also Moggi (1990), Wadler (1995), and Wadler (1997)). The purely functional Haskell programming language implements random numbers in just this way (Jones, 2003). The state of the random number generator is carried in the `IO` monad and this means that the type of our model needs to be modified to be a function to the state space under this monad. Iteratively evaluating the model is then done using the *bind* operation.

Anecdotally, however, the monad treatment is difficult for working programmers to grasp. One blog post puts it succinctly, "Monads are not trivial. If they were, there would not be so many tutorials and articles explaining them." (McBeath, 2008). They are not complicated, nevertheless this may be one reason for the relatively slow adoption of functional languages in scientific computing and industry. For simplicity therefore, in what follows we sacrifice a degree of elegance and adopt the engineering approach of augmenting the model and its state to include a random number generator.

2.2.3 Loops and Undefined Behaviour

It is quite possible that the output of a model is not defined for a given (t, x) input. The typical way this can happen is if the model iteratively computes some function until a condition is met. If the condition is never met, the iteration continues indefinitely and no output is ever produced. This might happen because of a bug or error in the the model. It might also happen simply because the model is incomplete or that there is some limitation in the underlying numerical routines. Indeed the model may simply be undefined for some otherwise valid input. In any case it is still of interest to be able to represent models exhibiting this behaviour in order to understand where they are valid, what errors may exist and so forth.

To account for the possibility of this kind of non-determinism, we consider that the model is not a total but a *partial function*; the model has the type, $\mathrm{M} : \mathbb{T} \to \mathbb{X} \rightharpoonup \mathbb{X}$. In the discussion that follows, the concept of propagator inherits this partial character.

2.3 Language Constructs

We now proceed to generalise the notion of propagator to permit the expression of implementation of models more sophisticated than systems of ordinary differential equations. A propagator is a partial function that maps the state space of the model into itself,

$$\varphi : \mathbb{X} \rightharpoonup \mathbb{X} \tag{2.8}$$

It is a partial function for the same reason that the underlying model (cf. Equation 2.1) is a partial function: there may be regions of state space for which it is undefined. Unlike the model that it implements, the propagator lends itself to a homogeneous description of the system. In other words it has only one parameter and its domain and range coincide. This type is convenient because it allows for unfettered composition to create new and different propagators out of simpler ones.

One could object on the grounds that the current time is an important feature of the state of the system and that in the homogeneous description of the system important information about time is lost. If time is not part of the structure of the state space and the propagator simply returns a new configuration of the system then no information is recorded about how much time has passed. This objection is easily addressed. One simply augments the state space by appending the elapsed time,

$$\begin{aligned} \mathbb{X}' &= \mathbb{X} \times \mathbb{T} \\ \mathbf{x}'_0 &= [\mathbf{x} \mid 0] \end{aligned} \tag{2.9}$$

Each propagator must simply update the elapsed time such that,

$$\begin{aligned} \pi_t\left(\mathbf{x}'\right) &= t \\ \pi_t\left(\varphi_h\left(\mathbf{x}'\right)\right) &= t + h \end{aligned} \tag{2.10}$$

where $\pi_t : \mathbb{X}' \to \mathbb{T}$ is the projection function that extracts the time component from the state.

2.3.1 Composition as Juxtaposition of Propagators

The exponential propagator has the convenient property that composition coincides with multiplication. That is,

$$(\varphi_a \varphi_b)(\mathbf{x}) = \varphi_a\left(\varphi_b\left(\mathbf{x}\right)\right) \tag{2.11}$$

where the left hand side matrix-matrix and matrix-vector multiplication and the right hand side is interpreted as function composition.

We take this equality as definitional: in this text, juxtaposing propagators as on the left hand side of Equation 2.11 means composition regardless of whether the propagators are of exponential kind. This affords us notational convenience. Extra symbols are unnecessary and it is possible to consider a composition of propagators as an entity on its own, without explicitly mentioning the state parameter, as in,

$$\varphi_c = \varphi_a \varphi_b \tag{2.12}$$

2.3.2 Iteration of Propagators

In some circumstances we wish to repetitively apply a propagator until some condition holds. We use square brackets with a subscript to denote this,

$$[\varphi]_p(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } p(\mathbf{x}, \varphi(\mathbf{x})) \text{ is true} \\ \varphi(\mathbf{x}) & \text{otherwise} \end{cases} \tag{2.13}$$

where p is a binary predicate, $p : \mathbb{X} \to \mathbb{X} \to \mathbb{B}$. If p evaluates to false, the result is the operation of the propagator. If p evaluates to true, the result is the identity. For iteration of the propagator until the condition is met we write, $[\varphi]_p^{\rightarrow}$. Since $id(x) = x$, we can say that if there exists some finite n for which the following holds,

$$[\varphi]_p^{\rightarrow}(\mathbf{x}) = [\varphi]_p^{n+1}(\mathbf{x}) = \varphi^n(\mathbf{x}) \tag{2.14}$$

This iteration construct is just the same as a *while* loop and the smallest such n is a count of how many times it has executed. If such an n exists, the computation terminates, otherwise it does not and the result is undefined.

2.3.3 Fixed-point Propagators

A useful special case of iteration is the (approximate) fixed point propagator. Given a function, $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, that encodes some notion of distance between two points in state space, we define the approximate fixed-point propagator as follows,

$$\varphi^\star = [\varphi]_{d,\varepsilon}^{\rightarrow} \tag{2.15}$$

This is used when the system should be advanced until it stabilises or converges to a certain state. In this converged state the following must hold,

$$d(\mathbf{x}, \varphi(\mathbf{x})) < \varepsilon \tag{2.16}$$

where $\varepsilon : \mathbb{R}_{\geq 0}$ is a sufficiently small real number.

As we shall see below when we consider optimisation and gradient descent, this kind of propagator is crucial for terminating the procedure. There is, however, no guarantee in general that a system will converge to such a fixed point.

2.3.4 Probabilistic Propagators

The final primitive enables probabilistic selection from a finite set of alternative propagators, $\Phi = \{\varphi_i\}$. Given a probability distribution, p_Φ, over this set, the stochastic propagator,

$$\varphi(\mathbf{x}) = \sigma\left(p_\Phi, \Phi, \mathbf{x}\right) \tag{2.17}$$

selects a propagator $\varphi \in \Phi$ according to the probability given by $p_\Phi(\varphi)$.

Recalling the discussion in Section 2.2.2, we point out that in order to support random selection of propagators as in Equation 2.17, we have two choices. We can consider that this kind of propagator is not a (partial) function from the state space to itself but rather to a distribution over the state space. Doing so would complicate the exposition so we instead consider only implementations that rely on pseudo-random number generators. We consider the pseudo-random number generator to be a part of the model and include its internal state in the models' state space. This is the reason for including the state explicitly in the function signature.

A randomly permuted propagator is one where each element of the set Φ is applied in a random order. The order is chosen by shuffling the elements according to the distribution p_Φ. A common choice is simply the uniform distribution to achieve a fair shuffling, but we account for the possibility of bias. The selection of an element of Φ is done by using the pseudo-random number generator to sample from the distribution p_Φ.

We construct a randomly permuted propagator in the following way. Let $\Phi_1 = \Phi$ and $p_{\Phi_i} = p_\Phi$. We then define the sequence of propagators,

$$\varphi_i\left(\mathbf{x}\right) = \sigma\left(p_{\Phi_i}, \Phi_i, \mathbf{x}\right) \tag{2.18}$$

where the sets are constructed by removing those elements that have already been used,

$$\Phi_i = \Phi_{i-1} \setminus \{\varphi_{i-1}\} \tag{2.19}$$

and the probability distributions are renormalised at each step,

$$p_{\Phi_i}(\varphi \in \Phi_i) = \frac{|\Phi_{i-1}|}{|\Phi_{i-1} - 1|} p_{\Phi_{i-1}}(\varphi) \tag{2.20}$$

For the common case where p_Φ is the uniform distribution, we write,

$$\underset{\sim}{\Phi} = \prod_{i=1}^{|\Phi|} \varphi_i \tag{2.21}$$

2.4 Examples

2.4.1 Recurrent Models

The models for which it is simplest to derive a propagator are discrete recurrent models. This is because the propagator itself is a recurrence relation, giving a position in state space in terms of some position in the past. We can directly define propagators such as this one, for the Hénon map (Hénon, 1976),

$$\varphi(\mathbf{x}) = \begin{bmatrix} 1 - ax_1^2 + x_2 \\ bx_2 \end{bmatrix}$$

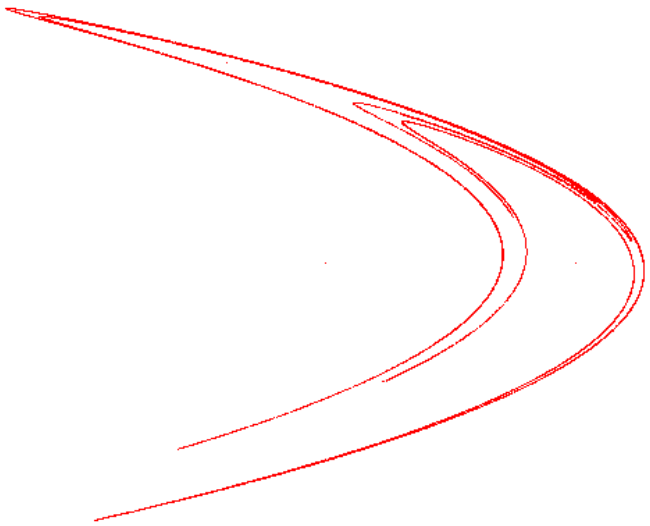


Figure 2.1: The Hénon map, simulated for 25,000 iterations or time steps with $a = 1.4$ and $b = 0.3$, simulated MOIS (Bucher et al., 2014), a general propagator-centric modelling software.

2.4.2 Pre-computation

Another simple kind of propagator is pre-computed or pre-measured. Suppose that a system depends exogenously on some quantity such as the level of the tide in a particular

place. It might be possible to model this quantity on its own, but perhaps it is expensive to do so, in which case it should be simulated only once and the results stored. Or it might not be known how to accurately model it, or inconvenient, or peripheral to the problem at hand but nevertheless possible to measure it accurately enough in nature.

It is a simple matter to consider such sets of pre-computed or measured data as models and to construct a propagator so that they may later be combined with other models more central to the object of study. Such data sets can be considered as sequences of tuples, $(t, \mathbf{x})$ and made homogeneous as in Equation 2.9.

A simple propagator for such a data set simply treats it as a step function. This is most easily written in code,

```
dataprop :: [X] → H → X → X
dataprop data h x = if time next ≥ h + time x then next else x
  where
    -- assume time is the first coordinate
    time = head
    -- not especially efficient way to find the next data point
    next = head $ filter (λd → time d > time x) data
```

The propagator is then implemented as a partial application, `phi = dataprop dataset` and propagators as step functions obtained by further applying it to a time-interval argument. This is not an especially efficient implementation, in particular on each application of the propagator it traverses the entire data set looking for the first entry where the time is greater than the current time, but it is sufficient to illustrate the idea.

2.4.3 Numerical Integrators

2.4.3.1 The Forward Euler Method

Suppose that a model has been formulated as a homogeneous initial value problem,

$$\dot{\mathbf{x}} = f(\mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{2.22}$$

and assume for the moment that $\mathbf{x}(t)$ is continuous and infinitely differentiable on an interval of interest, $[t,\ t+h]$.

This class of model is often formulated directly but also arises as a consequence of models posed in energy terms via the Euler-Lagrange equations or Hamilton's

equations (Goldstein, 1980). An exact solution is rarely possible, but expanding in Taylor series about $\mathbf{x}$ a first-order approximation is immediately obtainable:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) + \mathcal{O}(h^2) \tag{2.23}$$

which can be seen to have the form of a propagator,

$$\varphi_h(\mathbf{x}) = \mathbf{x} + h f(\mathbf{x}) \tag{2.24}$$

where terms second order and higher in h have been dropped. Of course this is immediately recognisable as the forward or *explicit Euler* numerical integration method (Teukolsky et al., 2007; Hairer et al., 1996), which is about the simplest possible of such methods.

2.4.3.2 The Backward Euler Method

It is well known that the explicit Euler integrator has poor stability properties (Teukolsky et al., 2007; Hairer et al., 1996). By observing that $\lim_{h\to 0} \dot{\mathbf{x}}(t+h) = \dot{\mathbf{x}}(t)$ and reasoning analogously to the previous Section 2.4.3.1, a closely-related method is obtained,

$$\varphi_h(\mathbf{x}) = \mathbf{x} + h f(\varphi_h(\mathbf{x})) \tag{2.25}$$

which uses the rate of change at the *end* of the time step rather than the beginning. It is an implicit expression for the result and must be solved by iterative methods, for example Newton's method (Teukolsky et al., 2007). For this reason is called the backward or *implicit* Euler method. It is much more stable when applied to certain problems than the explicit version. Though Equation 2.25 is implicit, it is still possible to write a propagator to approximate it to arbitrary accuracy, and so the method is still first order.

The implicit Euler method can be implemented as a propagator as follows with some conditions on f such as being differentiable in the neighbourhood of $\mathbf{x}$. Let e_h be the forward Euler propagator from Equation 2.24. We will use this to make an initial guess at the correct value for $\varphi_h(\mathbf{x})$:

$$\mathbf{x}_0 = e_h(\mathbf{x}) \tag{2.26}$$

Now, rearranging Equation 2.25, let,

$$g(\mathbf{x}_{k+1}) = \mathbf{x}_{k+1} - \mathbf{x}_k - h f(\mathbf{x}_{k+1}) = \mathbf{0} \tag{2.27}$$

Netwon's method gives us the approximation,

$$\nu(\mathbf{x}_k) = \mathbf{x}_{k+1} = \mathbf{x}_k - J_g^{-1}(\mathbf{x}_k) g(\mathbf{x}_k) \tag{2.28}$$

where $J_g(\mathbf{x})$ is the Jacobian matrix of g evaluated at $\mathbf{x}$.

All that is necessary to obtain an implicit Euler propagator is to repeat this propagator until a fixed point emerges, yielding,

$$\varphi_h = \nu^\star e_n \tag{2.29}$$

It is possible that this propagator will not terminate, however in practice that is rarely the case provided that h is sufficiently small. Because it is unusual to have the Jacobian matrix in analytical form and the cost of computing the Jacobian matrix at each step is large, a so-called quasi-Newton method (Broyden, 1967) is used.

2.4.4 The Symplectic Euler Integrator

Now we meet our first compound integrator, built out of embedded forward Euler methods, which will make it very easy to explicitly analyse. Suppose now for simplicity that $\mathbf{x} \in \mathbb{R}^2$. We can re-write Equation 2.22 as,

$$\dot{\mathbf{x}} = \begin{bmatrix} f(\mathbf{x}) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ g(\mathbf{x}) \end{bmatrix} \tag{2.30}$$

and this gives rise to two propagators analogously to Equation 2.24,

$$\begin{aligned} \varphi_h^{[f]}(\mathbf{x}) &= \begin{bmatrix} x_1 + h f(\mathbf{x}) \\ x_2 \end{bmatrix} \\ \varphi_h^{[g]}(\mathbf{x}) &= \begin{bmatrix} x_1 \\ x_2 + h g(\mathbf{x}) \end{bmatrix} \end{aligned} \tag{2.31}$$

working out what the composition produces we get,

$$\left(\varphi_h^{[g]} \varphi_h^{[f]}\right)(\mathbf{x}) = \begin{bmatrix} x_1 + h f(\mathbf{x}) \\ x_2 + h g\left([x_1 + h f(\mathbf{x}),\, x_2]^T\right) \end{bmatrix} \tag{2.32}$$

or in other words a solution that is explicit in one variable and explicit in the other. While it is still a first-order approximation to the true solution, it has very good stability properties (Hairer et al., 1996), much better than the explicit Euler method without the computational expense of the iterative methods required for the implicit Euler method. This is called the semi-implicit or *symplectic* Euler integrator. It is called semi-implicit because in the calculation of the second coordinate, the system is already advanced in the first coordinate.

2.4.5 Splitting and Composition

Equation 2.30 is an example of a split model of the more general form

$$\dot{\mathbf{x}} = \alpha(\mathbf{x}) + \beta(\mathbf{x}) \tag{2.33}$$

We now consider the relationship of the propagator for the combined function, $\varphi^{[\alpha+\beta]}$ with those for the individual ones, $\varphi^{[\alpha]}$ and $\varphi^{[\beta]}$. This is a broad topic considered in depth elsewhere (McLachlan, 1995; Hairer et al., 1996; Blanes et al., 2008) and we will just sketch a simple case here, though it can be shown that it is possible to obtain composition methods in this way up to very high order (Blanes et al., 2006).

There is a very simple result Hairer et al. (1996) that shows that composing these propagators, first doing one and then the other, gives results correct to first order. This echoes the symplectic Euler case but in a more general setting involving arbitrary splitting of differential equation systems. The result is as follows.

If $\varphi^{[\alpha+\beta]}$ is the propagator corresponding to Equation 2.33, and $\varphi^{[\alpha]}$ and $\varphi^{[\beta]}$ are the propagators corresponding each term on the right hand side, then the following holds:

$$\varphi_h^{[\alpha+\beta]} = \varphi_h^{[\beta]} \varphi_h^{[\alpha]} + \mathscr{O}(h^2) \tag{2.34}$$

Let us expand the value after the application of the first propagator in Taylor series and denote it as,

$$\mathbf{x}' = \varphi_h^{[\alpha]}(\mathbf{x}) = \mathbf{x} + h\,\alpha(\mathbf{x}) + \mathscr{O}(h^2) \tag{2.35}$$

Similarly, the final value after application of the second propagator is given to first order by,

$$\mathbf{x}'' = \varphi_h^{[\beta]}(\mathbf{x}') = \mathbf{x}' + h\,\beta(\mathbf{x}') + \mathscr{O}(h^2) \tag{2.36}$$

The value for $\beta(\mathbf{x}')$ is given by $\beta(\mathbf{x})$ plus terms themselves involving h, so,

$$\mathbf{x}'' = \mathbf{x} + h\,\alpha(\mathbf{x}) + h\,\beta(\mathbf{x}) + \mathscr{O}(h^2) \tag{2.37}$$

which can be immediately identified as the first order Taylor expansion for the result of applying the combined propagator.

This very basic result, nearly trivial, is very useful. It provides a bound on the accuracy of composition methods. It also shows that it is possible to have a composition method with "known properties" and what this means.

Interestingly the above theorem works in reverse as well. Under similar conditions, a propagator that is exact in composition can be approximated to first order (at least) by

the propagator that corresponds to their models acting simultaneously. To the extent that "acting simultaneously" means "could be implemented in parallel," this suggests that parallel approximation of serial programs are possible with known error, at least in these circumstances restricted to continuous and differentiable functions.

2.4.6 Equilibrium and Optimisation

Some models are not concerned with how a system changes in time at all. Rather they assert that the system, if it is out of equilibrium, will move rapidly to equilibrium. Mechanical models of epithelial tissues represented as a graph Nagai et al. (2001) and Farhadifar et al. (2007), for example, can easily be shown to have this property. The argument is as follows.

A perturbation to a patch of epithelial tissue, while it may cause a reconfiguration of the cells, will not cause it to remain in motion for an appreciable time. This is because the ratio of inertial to viscous forces, a quantity known as the Reynold's number (Reynolds, 1883), is characteristically low. In such systems, kinetic energy is removed quickly by friction.

The tissue is modelled with a potential function of the vertices where cell-edges come together, $U(\mathbf{q})$. In general, its motion will be described by the Euler-Lagrange equations (Goldstein, 1980) with an external velocity-dependent dissipative force $\mathbf{Q}(\dot{\mathbf{q}})$,

$$\frac{\partial L}{\partial \mathbf{q}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}} = \mathbf{Q} \tag{2.38}$$

where, $L = T - U$ and T is the kinetic energy of the system. If the systems is hypothesised to be at rest, then both T and its derivative with respect to velocity will be zero. The friction, Q, will also be equal zero for the same reason. Since U depends only on position, we are simply left with,

$$\frac{\partial L}{\partial \mathbf{q}} = -\nabla U = 0 \tag{2.39}$$

and the problem reduces to one of finding a minimum of a scalar function of the generalised positions.

2.4.6.1 Optimisation Methods

We have already seen an optimisation method in Section 2.4.3.2 for solving the implicit Euler integration scheme: Newton's method. Now we examine some others.

2.4.6.2 Coordinate Descent

The simplest, perhaps most naïve method for finding a local minimum of a function is to minimise each coordinate in turn. This is computationally cheap but does not work for many kinds of functions, particularly those that are not smooth. But because it is so simple it is a nice illustration of how fixed point propagators can be composed to construct a standard optimisation method.

Let $\varphi_{i,\eta}$ denote the function that for the ith coordinate, moves in the decreasing direction in proportion to the rate of change of the scalar potential in that direction. The η is the step size. This propagator is given explicitly by,

$$\varphi_{i,\eta}(\mathbf{q}) = \begin{bmatrix} q_1 \\ q_2 \\ \cdots \\ q_i - \eta \frac{\partial U}{\partial q_i} \\ \cdots \\ q_{n-1} \\ q_n \end{bmatrix} \tag{2.40}$$

Coordinate descent says that this must be continued until the ith coordinate is at a minimum, meaning $\frac{\partial U}{\partial q_i} = 0$ so what is required is a fixed point version of this propagator, $\varphi^{\star}_{i,\eta}$.

The prescription then is to do this same procedure for each coordinate in turn. So the propagator for one iteration of coordinate descent is given by,

$$\varphi_{\eta} = \prod_{i=1}^{n} \varphi^{\star}_{i,\eta} \tag{2.41}$$

The whole procedure is then repeated until the result converges (if it converges) to a minimum, so the entire coordinate descent procedure can be written as,

$$\varphi^{\star}_{\eta} = \left(\prod_{i=1}^{n} \varphi^{\star}_{i,\eta} \right)^{\star} \tag{2.42}$$

2.4.6.3 Gradient Descent

A slightly more sophisticated approach is, rather than minimising each coordinate independently, to change them all at the same time and move in the direction of steepest descent, the gradient of the potential,

$$\varphi_{\nabla,\eta}(\mathbf{q}) = \mathbf{q} - \eta \nabla U \tag{2.43}$$

giving one step of gradient descent.

As with coordinate descent, this procedure should be repeated until a fixed point is obtained, so we write $\varphi^{\star}_{\nabla,\eta}$ for it.

2.4.6.4 Stochastic Descent

Suppose we were concerned that the coordinate descent above that the order in which the coordinates were chosen would materially affect the result. To remove this potential bias, we might choose a random order on each iteration. This is easily represented by the stochastic propagator,

$$\underset{\sim}{\varphi^{\star}_{\eta}} = \underset{\sim}{\left\{\varphi_{i,\eta}\right\}^{\star}} \tag{2.44}$$

Stochastic gradient-descent, or on-line descent is a standard technique used in machine learning (Bottou, 1998). It is straightforward to describe this technique in our notation as well. Suppose that the potential can be written as the sum,

$$U(\mathbf{q}) = \frac{1}{n}\sum_{i=0}^{n} U_i(\mathbf{q}) \tag{2.45}$$

Such a potential can be split into the corresponding gradient descent propagators, for each summand,

$$U_i \quad \Rightarrow \quad = \varphi^{i}_{\nabla,\eta} \tag{2.46}$$

This compact notation says that the propagator does gradient descent ∇, with a step size or learning rate η, for the ith potential term. On-line descent says that we should iterate for each potential term in random order. Using our notation, this procedure is conveniently and compactly written as,

$$\underset{\sim}{\varphi^{\star}_{\nabla,\eta}} = \underset{\sim}{\left\{\varphi^{i}_{\nabla,\eta}\right\}^{\star}} \tag{2.47}$$

2.4.6.5 Quasi-Newton Methods

As was pointed out towards the end of Section 2.4.3.2, though it is straightforward to write Newton's method as the propagator ν, solving Equation 2.28 can be impractical because the Jacobian matrix is rarely available. Quasi-Newton methods (Broyden, 1967) approximate the Jacobian matrix at each step as well. There are several algorithms used for this, most commonly the classical Broyden-Fletcher-Goldfarb-Shanno (BFGS) method (Broyden, 1970a; Broyden, 1970b; R. Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) and somewhat more recently the Symmetric Rank One (SR1) method (Conn et al., 1991).

In the context of minimising scalar functions that we have been discussing, Equation 2.28 becomes,

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \eta_k B_k^{-1} \nabla U(\mathbf{q}_k) \tag{2.48}$$

where B_k is a symmetric matrix approximating the Hessian matrix of the function U, and η_k is the step size which, in general, should be allowed to vary for each iteration. The difference between the BFGS, SR1 and related methods is the specific way in which B_k is calculated and kept updated. In all methods, however, each iteration requires extra information not contained in the coordinates—the Hessian approximation and the step size.

To support these methods within our framework of propagators, we augment their domain and codomain. We locally augment the state space using a function to a space that includes an additional $n \times n$ matrix and a step size, $B = (\mathbb{X}, \mathbb{R}^n \times \mathbb{R}^n, \mathbb{R})$. The inverse of this function is clearly the projection onto $\mathbb{X}$, $\pi_{\mathbb{X}}$. So if $b : B \to B$ is the propagator that executes one step of the BFGS method, the propagator implementing the entire procedure is,

$$\varphi_{\text{bfgs}} = \pi_{\mathbb{X}} b^{\star} \pi_{\mathbb{X}}^{-1} \tag{2.49}$$

where $\pi_{\mathbb{X}}^{-1}$ is simply defined as,

$$\pi_{\mathbb{X}}^{-1}(\mathbf{x}) = [\mathbf{x}, \mid I_n, \eta_0] \tag{2.50}$$

for an initial step size η_0 and the $n \times n$ identity matrix.

2.4.7 The Loch Ness Monster

We next consider an interesting example from statistical mechanics, the treatment of simple reaction networks. We present this example as a brief tutorial in simple terms for clarity, inspired by the excellent textbook from Baez et al. (2012) and the thorough and rigorous treatment of Behr; Duchamp, et al. (2017). Indeed the title of this section has its origins in a presentation given by Behr some years ago in which the example reaction $A \to \emptyset$ was considered in detail. That example is reproduced below followed by an example of an equilibrium system where the reaction is made reversible.

This tutorial is intended to be merely illustrative and the model given here has severe limitations. The state-space of the model is infinite-dimensional. Each dimension is used to represent the probability of having a certain number of particles. There is no *a priori* limit on the number of these particles so there is no limit on the number of dimensions needed to represent their probabilities. We cannot store infinite-dimensional

counting vectors in computer memory, so we use a finite approximation. Truncating the state-space means that the probability of having more particles than can be represented, at any time, is negligibly small. This approximation limits the systems that can be modelled. A second important limitation pertains to systems with more than one species. Representing multiple species requires counting vectors for each one. Though it would be possible to concatenate the (already finitely approximated) vectors for each species to construct the model's state-space, doing so would complicate the exposition. We therefore restrict ourselves to systems involving only one species.

2.4.7.1 The Chemical Master Equation

The main purpose of this tutorial section is to demonstrate the software implementation of a propagator framework in the Haskell programming language for approximations of simple models expressed in terms of the Chemical Master Equation,

$$\frac{d\,|\psi(t)\rangle}{dt} = H\,|\psi(t)\rangle \tag{2.51}$$

This equation gives the rate of change of a system's state, $|\psi(t)\rangle$, as an infinite-dimensional vector describing the number of particles in the system upon which the Hamiltonian, H, acts to effect a change. The *ket* notation, $|\cdot\rangle$ indicates, essentially, that the state is a column vector. The conjugate, *bra* notation, $\langle\cdot|$ indicates a row vector. This is known as *bra-ket* or Dirac notation and it is used to distinguish infinite-dimensional vectors for ordinary, finite-dimensional vectors. The form of Equation 2.51 suggests that H is either a scalar or a square matrix, and it is, in fact, the latter.

2.4.7.2 Counting State Vectors

We now proceed to define $|\psi(t)\rangle$. In this setting, the system state is a count of particles. We consider only systems with a single kind of particle. For a system with one kind of particle, $|\psi(t)\rangle$, represents a time-varying vector in an infinite-dimensional real vector space. As such, $|\psi(t)\rangle$ is a function from the reals, representing time, to this counting space, $|\psi(t)\rangle : \mathbb{R} \to [0,1]^{\infty}$. Each coordinate of the vector represents the probability of having a certain number of particles, starting at zero. If the system contains n particles,

$$|\psi(t)\rangle = [Pr(n=0), Pr(n=1), Pr(n=2), \ldots]^T \tag{2.52}$$

For example, the state $[1,0,0,\ldots]^T$ indicates that there are no particles present and the state $[0,0,0,0,0,1,0,0,\ldots]^T$ indicates that there are five. The state $[0,0.5,0,0,0.5,0,0,\ldots]^T$ represents an even chance of finding the system with one or four particles.

The state vector is normalised such that,

$$\sum_{i=0}^{\infty}(|\psi(t)\rangle)_i = 1 \tag{2.53}$$

where the sum is taken over all components so that it can be interpreted as a probability distribution. The probability to have n, particles, $p(n)$, is simply the value of the nth component of $|\psi(t)\rangle$. It is defined for all non-negative integers n and it is a probability distribution because of the normalisation of $|\psi(t)\rangle$. This ability to represent a distribution for probabilities of having any given number of particles as a vector is the reason for using this kind of infinite-dimensional space rather than simply using, for example, an integer.

The solution to equation 2.51 is,

$$|\psi(t)\rangle = e^{tH}|\psi(0)\rangle \tag{2.54}$$

which, apart from specialised ket notation, is equivalent to Equation 2.4. Our goal is to construct the propagator, $\varphi_t = e^{tH}$, corresponding to Equation 2.5. This is challenging because of the infinite order of the state space. Our strategy is to approximate it finitely, and we begin by examining the nature of the Hamiltonian matrix operator.

2.4.7.3 Creation and Annihilation

Changes to the system are naturally expressed in the form,

$$A \xrightarrow{k} B \tag{2.55}$$

where some reagents A interact and produce some products B at the rate k. Underlying this representation are the concepts of creation and annihilation, some inputs to the reaction are annihilated and some outputs are created. We very loosely follow the presentation of Baez et al. (2012), who also highlight the intimate connection with Quantum Mechanics and the correspondence between the Master Equation and the Schrödinger Equation.

We will write $a^\dagger$ for the operator that creates a particle and a for the operator that destroys one. Both of these operators not only effect the creation or annihilation but count the ways in which it can be accomplished. Adopting the shorthand $|n\rangle$ for the state vector with a 1 in position n and 0 elsewhere, we write the intended effects of $a^\dagger$

and a as,

$$a^\dagger |n\rangle = |n+1\rangle \tag{2.56}$$

$$a|n\rangle = \begin{cases} n|n-1\rangle & n > 0 \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{2.57}$$

The creation operator is straightforward, there is a single way to create a particle, simply by adding one to the system, and the result is a new state with one more particle than there was before, $|n+1\rangle$. The annihilation operator is more subtle. If there are no particles, none can be removed, so the result is $\mathbf{0}$ — note not $|0\rangle$ since this not the state of having no particles but the number of ways no particles can be removed, in effect, a kind of extinction. If there is a non-zero number of particles there are as many ways as there are particles to remove one, with one fewer left over, hence $n|n-1\rangle$. The creation or annihilation of several particles are simply powers of these operators. A double creation operator is just $a^\dagger a^\dagger = a^{\dagger 2}$.

Let us be explicit about the matrix representation of these operators. After all it will be necessary to implement them. The creation operator, $a^\dagger$, in explicit (infinite) matrix form, is,

$$a^\dagger = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \\ 0 & 0 & 0 & 1 & 0 & \ddots \\ \vdots & & \vdots & & \ddots & \ddots \end{bmatrix} \tag{2.58}$$

It is easy to satisfy oneself that this matrix satisfies Equation 2.56.

In code, this matrix can be generated easily from a sparse association list for finite arbitrary size. We cannot generate infinite matrices as such, therefore we truncate the state space so that the matrix representation of the operator can be of finite size. Such truncated representations have been studied. Munsky et al. (2006) and later Burrage et al. (2006), for example, dynamically computes the truncation error and finitely expands the state space as required. Cao et al. (2016) employs reflecting boundaries on a truncated state space where increases in particle counts above the finite size become decreases, and they compute the error for this. Here, we are simply demonstrating a toy implementation of our propagator notation for some interesting particular cases, so we do not implement dynamically changing state dimension or reflecting boundaries or compute rigorous

error estimates. We do note that, owing to the unchanging state dimension, the reflecting boundary strategy appears most amenable to treatment in our framework.

This sparse representation is transformed to a dense matrix for pragmatic purposes because the matrix exponential function *expm* that we will eventually use requires a dense matrix representation. The code is as follows,

```
create :: Int → Int → Matrix Double
create n p = LD.toDense $ ((n-1, n-1), 0):m
  where m = [ ((j+p, j), 1) | j ← [0..(n-p-1)]]
```

The first argument `n` to `create` is the size of the matrix to produce, and the second one, `p`, is the power. Because we can calculate powers of these matrices directly, and this will be a frequent requirement, we optimise here instead of multiplying individual creation matrix operators together. This function produces a matrix of size `n` for $a^{\dagger p}$. The association list is prepended with a zero entry in the bottom right corner to establish the intended size of the matrix.

The annihilation operator can be handled in a similar way. It can also be explicitly represented as an infinite matrix,

$$a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 2 & 0 & 0 & \\ 0 & 0 & 0 & 3 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 4 & \\ 0 & 0 & 0 & 0 & 0 & \ddots \\ \vdots & & \vdots & & \ddots & \ddots \end{bmatrix} \tag{2.59}$$

Note that the case definition in Equation 2.57 follows directly from this matrix representation. Calculating $a|0\rangle$ by multiplication with this matrix gives the required $\mathbf{0}$ vector.

To represent the annihilation operator in code, similarly optimised for directly calculating powers of this matrix without multiplying them explicitly, we need to implement an auxiliary falling factorial function,

```
(!.) :: Int → Int → Int
n !. k = product [(n-k+1)..n]
```

Implementing a^p is now,

```
annihilate :: Int → Int → Matrix Double
annihilate n p = LD.toDense $ ((n-1, n-1), 0):m
  where m = [ ((j, j+p), fromIntegral (j+p !. p)) | j ← [0..(n-p-1)] ]
```

where the arguments have the same meaning as with `create`.

To construct a reaction from these operators, one might be tempted to simply write $a^{\dagger t}a^s$ where t and s are the number of particles created and destroyed, respectively. This would be wrong because of the counting behaviour of the annihilation operator. When applied to $|n\rangle$ it is wrong by an extra factor of $n(n-1)(n-2)\dots(n-s)$ so long as $n \geq s$. To correct for this, we subtract $a^{\dagger s}a^s$ to arrive at the reaction, ρ, which first counts the number of ways in which s particles may be consumed and if there are enough, creates t new ones,

$$\rho = k(a^{\dagger t} - a^{\dagger s})a^s \tag{2.60}$$

where k is the average rate at which the reaction occurs. This is the realisation in terms of creation and annihilation operators of the reaction that is normally written as,

$$sA \to tA \tag{2.61}$$

2.4.7.4 Reactions and the Hamiltonian

Reactions are now straightforwardly defined in code, as before with a certain finite matrix size, `n`, and a number of instances of particles that they create `t` or destroy `s` at a rate, `k`,

```
reaction :: Int → Int → Int → Double → Matrix Double
reaction n t s k = scale k (ct × as - cs × as)
  where
    ct = create n t
    cs = create n s
    as = annihilate n s
```

For a model with more than one reaction, the total effect is just the sum of their individual effects, and its Hamiltonian is written,

$$H = \sum_{\tau \in T} \rho_\tau = \sum_{\tau \in T} k_\tau (a^{\dagger t_\tau} - a^{\dagger s_\tau})a^{s_\tau} \tag{2.62}$$

where T is the set of reactions, and for each of them k_τ is the rate at which it happens, t_τ is the number of particles created and s_τ is the number annihilated.

If we specify reactions in code as a tuple of the form, `(t, s, r)`, in other words the parameters that we need to create an individual reaction, we can construct a finite representation of size `n` of the Hamiltonian from it,

```
hamiltonian :: Int → [(Int, Int, Double)] → Matrix Double
hamiltonian n rs = foldl1 (+) $ map(λ(t, s, k) → reaction n t s k) rs
```

In words, we create a reaction for each specification in the list, and then add them all together.

2.4.7.5 The Propagator

The propagator corresponding to the Hamiltonian, $\varphi_t = e^{tH}$, is nearly trivial to write in code,

```
propagator :: Double → Matrix Double → Matrix Double
propagator t h = expm (scale t h)
```

where, for clarity because of the syntax of the Haskell language, `h` is in fact the Hamiltonian, H, and t represents time. The function `expm` is from the *HMatrix* package (Ruiz, 2012) and computes a matrix exponential. (In this case, using a scaling and squaring technique with Padé approximation from Golub et al. (2012), though this is just an implementation detail. Numerical computation of matrix exponentials is not at all straightforward, see Moler et al. (2003).)

Finally we add a utility function that does the very straightforward computation conducting the evolution `n` times for an interval of `dt` and saves each step to a file on disk,

```
timeseries :: FilePath → Int → Double →
              Matrix Double → Matrix Double → IO ()
timeseries base n dt h x = mapM_ (λk → dump k (fromIntegral k*dt)) [1..n]
  where
    filename k = printf "%s_%04d.dat" base k
    dump k t   =
```

```
      saveMatrix (filename k) "%f" $ propagator t h × x
```

Note that because the problem admits an exact solution this function makes use of the fact that $e^{\alpha H} e^{\beta H} = e^{(\alpha+\beta)H}$ and produces the result for each time directly. In some instances this may be beneficial for reducing accumulated numerical error, leaning on the underlying implementation of the matrix exponential instead of explicit repeated application of the propagator. This approach is also computationally more expensive because a matrix exponential must be computed for each time-step.

Instead, accepting the risk of possible numerical errors, we use an implementation with a fixed propagator that is calculated only once. Not only is this more in the spirit of the framework that we have discussed so far, it is much faster since multiplying a matrix by a vector is far cheaper than computing a matrix exponential.

```
timeseries :: FilePath → Int → Double →
              Matrix Double → Matrix Double → IO ()
timeseries base n dt h x = dump n x
  where
    filename k = printf "%s_%04d.dat" base k
    prop     = propagator dt h
    dump 0 y = saveMatrix (filename n) "%f" y
    dump k y = do
      let y' = prop × y
      saveMatrix (filename (n-k)) "%f" y'
      dump (k-1) y'
```

2.4.7.6 Nessie

Using what we have created so far, here is a simple driver program to compute the dissipation reaction,

$$A \to \emptyset \tag{2.63}$$

where it is fed with an initial probability distribution of equal probability to have 10, 100, 200 or 300 particles. The size of the finite matrix representation is chosen to be slightly larger than the greatest number of particles that is expected.

```
main :: IO ()
```

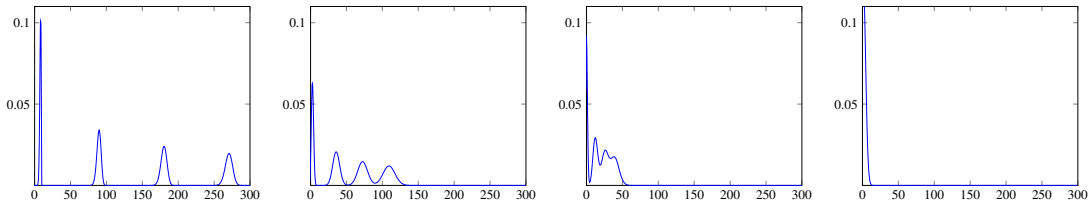


Figure 2.2: Probability distribution evolution for the dissipation reaction. In each plot, the vertical axis is probability and the horizontal axis is the number of particles. Starting from a distribution with a 25% probability of having each of 10, 100, 200 or 300 particles, the plot shows the distribution at $t = 0.1$, $t = 1$, $t = 2$ and $t = 4$.

```
main =
  let size = 330
      h = hamiltonian size [(0, 1, 1)]
      x = tr $ LD.toDense [((0,10), 0.25)
                          , ((0,100), 0.25)
                          , ((0,200), 0.25)
                          , ((0,300), 0.25)
                          ,((0,size-1), 0)
                          ]
  in timeseries "dissipation" 100 0.1 h x
```

The results are plotted in figure 2.2, and until the distribution converges to a Kronecker delta. The resemblance to the Loch Ness Monster is uncanny.

2.4.7.7 Equilibrium

Consider now the slightly more complicated system,

$$\emptyset \xrightarrow{k_0} A \tag{2.64}$$

$$A \xrightarrow{k_1} \emptyset \tag{2.65}$$

This is a reaction system for which the limit distribution, as $t \to \infty$, is known analytically to be Poisson with parameter $\lambda = k_0/k_1$ (Jahnke et al., 2007). To demonstrate this, we perform the experiment for both $\lambda = 1$ and $\lambda = 25$. The initial configuration of the system will be a Kronecker delta shifted to 100, $p(x) = \delta[100 - x]$, in other words certainty that there are 100 particles. The results are shown in Figure 2.3.

The driver code is,

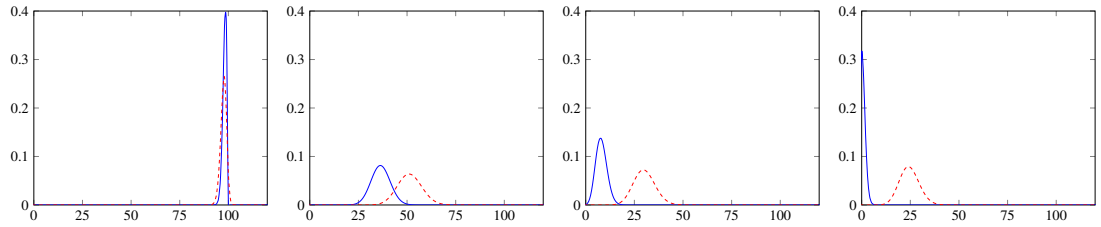


Figure 2.3: Probability distribution evolution for the equilibrium reaction set. In each plot, the vertical axis is probability and the horizontal axis is the number of particles. Starting with a distribution for 100 particles, the plot shows the distribution at $t = 0.1$, $t = 10$, $t = 25$ and $t = 50$. In each plot, the blue (solid) line is $k_0/k_1 = 1$ and the red (dashed) line is $k_0/k_1 = 25$.

```
main :: IO ()
main =
  let size = 600
      k0 = 25
      k1 = 1
      h = hamiltonian size [(1, 0, k0), (0, 1, k1)]
      x = tr $ LD.toDense [((0,100), 1), ((0, size-1), 0)]
  in timeseries "equilibrium" 500 0.01 h x
```

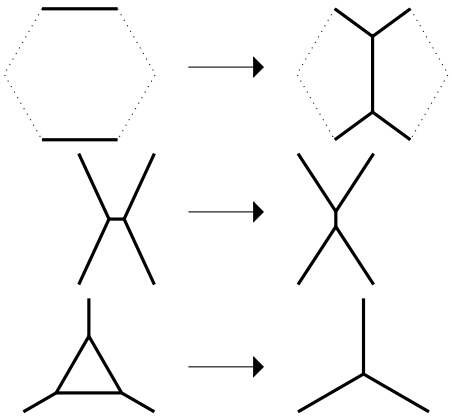
2.4.8 Farhadifar's Vertex Model

Finally, to illustrate the utility of our propagator notation in the setting of a more complex model, let us consider the vertex model of epithelial tissue presented in Farhadifar et al. (2007). It accurately predicts the topology of the embryonic wing disk of *drosophila*, a kind of fruit fly, and we briefly describe the model here.

In Farhadifar's model, a tightly packed single layer sheet of cells is represented as a graph. The edges in the graph are the edges of cells, and the vertices where these meet have a valence of three for reasons of mechanical stability (C. S. Smith, 2015). The cells themselves are chords in this graph. The model is made of two fundamentally different processes: the movement of vertices in space under the influence of mechanical forces due to a potential, and topological transitions which happen under certain conditions.

The topological transitions that are possible in such an environment have long been

known (C. S. Smith, 2015; Weaire et al., 2009) and are,



Division (2.66)

Migration (2.67)

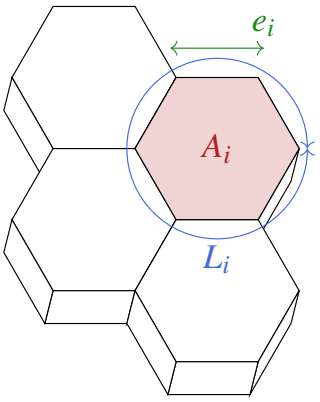
Extrusion (2.68)

Under proliferative conditions, division is performed at the beginning of each simulation time step. It is the primary source of perturbation in the system. The division rule says that a cell is chosen and two of its edges are bisected and joined. Migration, or edge-swapping, is performed when the cells on the left and right of the short edge are advancing towards each other and become sufficiently close to touch. Extrusion, the removal of triangles, is performed whenever it is possible. It may happen as a consequence of either of the two other transitions.

The potential governing the motion of the vertices between topological changes is,

$$U = \sum_{i \in \text{cells}} \frac{K}{2} (A_i - A_0)^2 + \sum_{i \in \text{edges}} \Lambda e_i + \sum_{i \in \text{cells}} \frac{\Gamma}{2} (L_i - L_0)^2 \tag{2.69}$$

The meaning of the main symbols is shown graphically in the figure at right. The potential contains a quadratic term relating to the area of each cell. The area if the ith cell is A_i. Cells have a preferred area, A_0, and experience an inward or outward force according to whether they are stretched or compressed. The model parameter K determines how strong this effect is. Edges have a cost per unit length, Λ, and the linear term acts to minimise their length, e_i, thus modelling the elasticity of the cell membrane. An addition to this model not present in the otherwise similar one of Nagai et al. (2001) is a third, quadratic term in the perimeter of the cell, modelling contractility or surface tension. L_i is the perimiter of the ith cell and, similarly to the case with area, L_0 is a preferred perimeter. Γ governs the strength of this effect.



In words, the simulation proceeds as follows. At the beginning of each time-step a cell is chosen to divide. Any three-sided cells present are removed. At this point the

tissue will be out of equilibrium, so the remainder of the procedure is repeated until it reaches equilibrium. A gradient descent is performed. If applicable, any migration, or edge-swap transitions are performed. If this has produced any three-sided ells are produced in the process, they are removed.

The equivalent description of the dynamic tissue simulation in terms of propagators is both unambiguous, more compact and perhaps more easily understandable. Let $\varphi_{\curlyvee}$, $\varphi_{\times}$ and $\varphi_{\perp}$ correspond to the division, migration and extrusion transitions respectively. As in Equation 2.43, $\varphi_{\nabla,\eta}$ performs one step of gradient descent. The propagator for one simulation step is then given by,

$$\varphi = \left(\varphi_{\perp}^{\star}\varphi_{\times}^{\star}\varphi_{\nabla,\eta}\right)^{\star}\varphi_{\perp}^{\star}\varphi_{\curlyvee} \tag{2.70}$$

An alternate implementation that produces comparable results is present in the Chaste software package (Mirams et al., 2013). This is used, for example, in Waites; Cavaliere, et al. (2018) and forms the basis for Chapter 3. It uses the same potential, and the same topological transitions, but operates differently. Rather than conducting a gradient descent, each vertex is moved independently, in random order. Essentially this is a stochastic splitting and composition method such as we have described above. If we write $\varphi_{\nabla,\eta}^{i}$ for the gradient descent propagator for the ith vertex, then this version of the simulation is,

$$\varphi = \left(\varphi_{\perp}^{\star}\varphi_{\times}^{\star}\left\{\varphi_{\nabla,\eta}\right\}\right)^{\star}\varphi_{\perp}^{\star}\varphi_{\curlyvee} \tag{2.71}$$

The difference between the simulations is very clear simply by inspection.

2.5 Conclusions

We have defined, starting from long-established practice in dynamical systems and numerical integrators, a notation of propagators for describing heterogeneous simulation algorithms. Existing practice has been extended with a concept of iteration and probabilistic selection of integrators from the members of a set. We have illustrated this notation by showing examples from discrete mathematics and some simple numerical methods in widespread use. We have further shown that, with our extensions, it is possible to express optimisation of optimisation techniques in a novel, succinct and unambiguous way. We have considered a toy implementation of the Chemical Master Equation, applied it to a dissipative system, and examined the effects of finitely approximating matrix operators in numerical approximation of non-dissipative systems.

Finally, we have shown how this notation can be used to clearly and unambiguously describe two different implementations of a fairly complex model, Farhadifar's vertex model for epithelia.

Chapter 3

Path Entropy

Abstract. We present *path entropy*, an information-theoretic measure that captures the notion of patterning due to phase separation in organic tissues. Recent work has demonstrated, both *in silico* and *in vitro*, that phase separation in epithelia can arise simply from the forces at play between cells with differing mechanical properties. These qualitative results give rise to numerous questions about how the degree of patterning relates to model parameters or underlying biophysical properties. Answering these questions requires a consistent and meaningful way of quantifying degree of patterning which we define. It is a resolution-independent measure that is better suited than image-processing techniques for comparing cellular structures. We show how this measure can be usefully applied in a selection of scenarios from biological experiment and computer simulation, and argue for the establishment of a tissue-graph library to assist with parameter estimation for synthetic morphology.

3.1 Introduction

One of the major mechanisms for understanding tissue development is adhesion-mediated sorting of cell mixtures into homotypic groups which was discovered by Steinberg in the 1960s (Steinberg, 1962). Interest in this phase separation mechanism has recently surged, partly because of its ability to create synthetic biological pattern-

ing mechanisms (Cachat et al., 2016) and partly because it has been found to drive events critical to the formation of organoids from stem cells (Unbekandt et al., 2010; Lefevre et al., 2017), making the process relevant to biotechnology as well as to basic development.

These investigations in experimental and synthetic biology have been paralleled by the development of analytic and computational models to explain pattern development. The first class of these are reaction-diffusion systems such as those of Turing (Turing, 1952) and Gierer (Gierer et al., 1972) in which a slowly diffusing activator molecule activates its own synthesis and also the synthesis of a rapidly diffusing inhibitor molecule. In such a system, small random asymmetries lead to slightly elevated production of activator morphogens and become centres of activator production and inhibit nearby sites from doing the same. The result is a field with separated spots, patches or stripes of high activator expression, which can be modelled for a two-component fluid system by the Cahn-Hilliard equation (Cahn et al., 1958).

The second class of model is discrete, and patterning emerges from the mechanical properties of the cells themselves: cell-cell adhesion, contractility, and the balance between cell surface area and volume. In this class are the Cellular Potts model (Graner et al., 1992) and the model of Newman et al. (Sandersius et al., 2011), in which motion takes place on a mesh of a scale much smaller than a cell, and Vertex models (Nagai et al., 2001; Farhadifar et al., 2007), in which the the system is represented as a dynamic and irregular mesh where polygons correspond directly to cells. Recently, analytic results have become available (Staple et al., 2010; Magno et al., 2015) that predict cell shapes produced by both numerical simulations and models in a homogeneous setting and they have been demonstrated (Osborne et al., 2017) to produce phase separation in simulation in a heterogeneous setting.

Against this background, there is a dearth of tools for comparing data produced by each of these disparate methods. Qualitatively, snapshots of tissues undergoing phase separation in simulation (Magno et al., 2015; Osborne et al., 2017) look similar to those produced experimentally by engineering cells with different levels of cadherin molecules (Cachat et al., 2016). In both cases the mechanism is understood to be Steinbergian differential adhesion, but there commonly used methods for quantitative techniques on epithelial sheets are mainly concerned with polygon distributions (Sánchez-Gutiérrez et al., 2015) or structural motifs (Vicente-Munuera et al., 2017) and are not straightforwardly extended to a setting with multiple cell types.

Graph-based distance, or graph similarity measures are well known. Eschera and

Fu (Eshera et al., 1984) define a distance between attributed feature graphs extracted from images in terms of transformations required to derive one from the other. Others such as Bunke and Shearer (Bunke et al., 1998) define a distance (in fact, a metric) in terms of the size of the maximal common subgraph. Measures of these types do not, however, contain any intrinsic notion of pattern or information so do not adequately capture these higher-level concepts. They are, in a sense, overspecific.

Shannon's entropy (Shannon, 2001) has proven difficult to extend to two or more dimensions in a meaningful way. The fundamental problem is that entropy depends fundamentally on the underlying probability distribution over *some* set of possibilities, but there is no unique way to decide *which* set is appropriate. It is an extrinsic anthropomorphic concept, not an intrinsic property of the system (Jaynes, 1965). Information-theoretic measures for images are known, but they are typically constructed on the probability distribution of pixel values in an image (Tsai et al., 2008; Gonzalez et al., 2004; Mangin, 2000), essentially transforming a two-dimensional problem into one dimension, sacrificing spatial structure in the process.

The Maximum Entropy technique (Skilling et al., 1984), widely used in image reconstruction from partial data, treats an image as a two dimensional structure, but is necessarily sensitive to image resolution. Likewise, other measures such as by Rubner et al. (Rubner et al., 2000) and the vast literature on distances between images for retrieval purposes do encode something of the information content, this is relative to the image resolution. For that reason, without some kind of pre-alignment such as with Cuturi & Doucet's technique of fast computation of Wasserstein Barycentres (Cuturi et al., 2014), they are not directly applicable to the task of comparing tissue examples from vastly different sources — experimental imagery on the one hand and simulation data on the other. A similar criticism can be made of Larkin's delentropy measure (Larkin, 2016) (however, see section 2 of Larkin's paper for an extended discussion of information-theoretic measures of images).

In this chapter, we provide such a method by defining a family of resolution-independent entropy measures on graphs that captures the different patterns observed throughout the literature on phase separation in cellular tissues. We choose to frame the measure in terms of graphs not only because the cell-cell contacts of epithelial and other biological tissues are intrinsically graph-like (Escudero et al., 2011), but because it is independent of the scaling or resolution of imagery. The property of resolution-independence is important because it allows comparison across different experiments, both *in-vitro* and *in-silico*. Using this measure, it is possible to answer such

salient questions as how quickly a pattern forms when starting from a random tissue or to meaningfully compare the degree of patterning observed in different numerical or wet-lab experiments. This capability enables workflows in synthetic mammalian biology where the goal is to engineer cell lines that will produce these kinds of patterns. Whether and to what extent the desired pattern is achieved can be consistently measured, and this information fed back into the system as the genome, host environment or external stimulus is adjusted.

3.2 Mathematical Preliminaries

We will use some concepts from graph theory and from information theory and probability. We assume a basic level familiarity with these on the part of the reader. Nevertheless, we review some key definitions and clarify the notation that we use throughout.

A set, X is a collection of elements. The number of elements in the set, its cardinality, is written as $|X|$. If another set Y is a subset of X, written $Y \subseteq X$ then the chance of choosing an element x of X uniformly at random and finding that it is also an element of Y is $Pr(x \in Y) = \frac{|Y|}{|X|}$.

A partition of a set, is a set of non-empty subsets of X called $\{Y_i\}$, such that each element in X is in exactly one of the Y_i. A partition gives rise to a probability distribution, which has the property that,

$$\sum_i Pr(x \in Y_i) = \sum_i \frac{|Y_i|}{|X|} = 1 \tag{3.1}$$

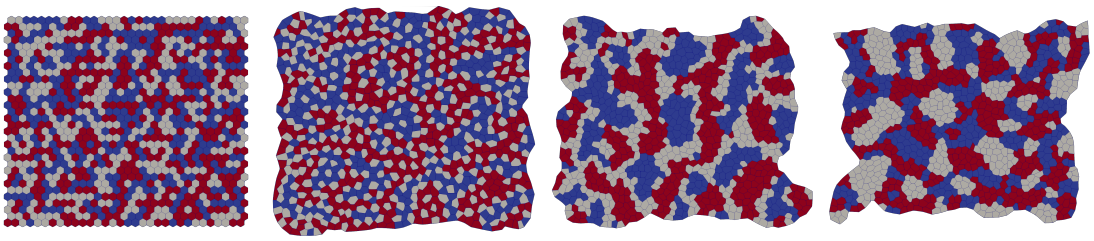
We will use one fact about Markov processes in this chapter. Suppose that the Y_i are possible states of a system in the state-space X and that there is a certain chance, q_{ij} that, being in state Y_i now at the next instant it will be in state Y_j. Since we require that the system is always in *some* state, it must be that,

$$\forall i, \quad \sum_j q_{ij} = 1 \tag{3.2}$$

A matrix, q_{ij}, with this property is called *stochastic*.

The Cartesian product of two sets, $X \times Y$ is the set of pairs $(x \in X, y \in Y)$. If both sets are the same, this is also written as X^2 and analogously for higher powers.

A directed graph, G, consists of a set of vertices, V, also called nodes, and a set of edges that connect the vertices, $E \subset V^2$. A path of length n on the graph is a sequence of vertices, $(v_0, v_1, \ldots, v_n)$ such that $(v_i, v_{i+1}) \in E$ for $0 \leq i < n$. We take the special



(a) Initial conditions (b) Case I (c) Case III (d) Case IV

Figure 3.1: A selection exemplar simulated tissue configurations for three cell types in equal proportion. At left, randomly distributed populations on a regular hexagonal lattice (typical initial conditions for simulation). The others the resulting configuration after some elapsed time for different costs of heterotypic and homotypic edges between cells. In particular, the cost of a heterotypic edge with a white cell increases from left to right, and the cost of a homotypic edge between white cells decreases. The precise meaning of Case I through Case IV is explained in Section 3.11.

case of zero-length paths to be simply the set of vertices itself. Let us write $S_n(G)$ for the set of all paths of length n from the graph, G.

A *graph invariant* is a quantity that depends only on the structure of the graph itself and not any representation or labelling. In particular it is a quantity that is invariant under graph isomorphism.

Let C be a set of colours and $\chi : V \to C$ be a function that maps vertices to colours. A coloured graph, (V, E, χ) is a graph together with such a function. Note that χ induces a partition on G when applied to each vertex. This partition map groups vertices together by colour.

3.3 Path Entropy

To motivate our pattern complexity measure more concretely, let us consider some exemplar simulated tissues, shown in Figure 3.1. These consist of three kinds of cells, represented as different colours, in equal proportion. The qualitative difference between each of the images is intuitively clear, from no discernible pattern, to a kind of quasi-uniform distribution of white cells, long, thin stripes and round patches reminiscent of the “dappling” calculated by hand by Turing. We seek a measurement that can be made on these that is able to distinguish them.

We choose to define this measure on the coloured adjacency graph of cells as

opposed to, for example, an image of the tissue as in the approach taken, for example, in (Larkin, 2016). The reason for this choice is that when calculated on the graph, the measure is resolution independent. It can be applied equally well to simulation data that has no intrinsic notion of image or resolution or to processed output from experimental imagery of cell colonies or epithelial sheets. As an important goal is to be able to compare data from different sources this property is important.

Let us proceed as follows. The entities of interest are cells so let us say that V corresponds to the set of cells in a given tissue. Further, let E be the edges, the adjacencies between cells. The patterns of interest are meaningful in terms of different kinds of cells so let the colours, C, correspond to the kind. For the purposes of this chapter we are concerned with the resulting coloured graph which we call the *adjacency graph of cells*.

Intuitively, a pattern is found in the sequence of colours extending out in one direction or another from a given point in the tissue. To capture this, we lift the colouring function from operating on vertices, to operating on sequences of vertices, or paths, $\chi_n : S_n(G) \to C^n$ for a given path length, n. As with χ, the χ_n induces a partition of $S_n(G)$: paths with the same colour sequence get into the same class.

We use this partition to obtain a probability distribution over C^n,

$$p_n(G)(s) = \frac{|\{\sigma \in S_n(G), \chi_n(\sigma) = s\}|}{|S_n(G)|} = \frac{|\chi_n^{-1}(s)|}{|S_n(G)|} \tag{3.3}$$

where $s \in C^n$. Where there is no risk of confusion or ambiguity, we will write $p_n(s)$ in place of $p_n(G)(s)$ from now on.

Definition 3.1

Given a coloured graph, $G = (V, E, \chi)$ and the probability distribution over colour sequences given by Equation 3.3, we define the *n-th order Path Entropy* on the graph to be the Shannon Entropy of this distribution:

$$E_n(G) = -\sum_{s \in C^{n+1}} p_n(s) \log_2 (p_n(s)) \tag{3.4}$$

As with the probability distribution, we write simply E_n in place of $E_n(G)$ where there is no risk of confusion.

Note that though the motivation is a measure on planar graphs representing epithelial sheets, there is nothing in this formulation that presupposes such a restriction. The family of entropy measures is equally well defined on graphs that embed into three or higher dimensional spaces.

3.4 Generalisation to Motifs

The foregoing is concerned with paths only, one-dimensional sequences of vertices. There is evidence that it may be fruitful to consider two dimensional motifs, or graph fragments (Vicente-Munuera et al., 2017). The approach given here can be straightforwardly applied to motifs. The general pattern for defining an entropy on a graph is to come up with a partition map and use the probability distribution that arises from that to get an entropy (Dehmer et al., 2011). A set of motifs induces a partition on the graph: the set of sets of subgraphs matched by each motif. Indeed a path of length n is simply a special kind of motif.

In order to deal with coloured graphs, or heterogeneous tissues, the matching function is simply lifted to a form that distinguishes differently coloured motifs as opposed to the purely structural ones considered by Vincente et al. This is precisely analogous to the coloured paths that we have used above. The corresponding notion of *Motif Entropy* follows directly.

3.5 Computational Complexity of Path Entropy

The steps required to calculate E_n, following directly from the definition in Section 3.3, are as follows.

A. We begin by enumerating of paths of length n, $S_n(G)$. This can be accomplished with a depth-first search to depth n for each vertex. Fortunately, though the number of paths can be very large, $|V|^{n+1}$ for a complete graph, we do not need to store the paths themselves: we can simply proceed to the next step and count occurrences of each colour sequence. Naturally we need to produce each path, so we must have time complexity of $\mathcal{O}(k|S_n(G)|)$ where k is a factor describing the complexity of producing a single path. This method time complexity in the worst case of $\mathcal{O}(|V|^{n+1})$ (Stickel et al., 1985) for a complete graph, and because paths can be produced incrementally during the search, k must be no more than a constant. For planar graphs of the kind considered here, where average degree $\langle d \rangle \approx 6$ (Sánchez-Gutiérrez et al., 2015), the situation is somewhat better, with time complexity of $\mathcal{O}(|V|\,\langle d \rangle^n)$. The depth-first search has space complexity of $\mathcal{O}(|V|)$ to keep track of each vertex visited.

B. For each path, $\sigma \in S_n(G)$, we compute its colour sequence, $\chi_n(\sigma)$, and count

the occurrences of each sequence. This requires visiting each vertex $v \in \sigma$ and computing $\chi(v)$. The time complexity is therefore $\mathcal{O}(|S_n(G)|)$, just as for the previous step. An upper bound on the space complexity can be obtained by supposing that all possible colour sequences occur. This is certainly the case for small numbers of colours and short paths such as we consider here. In this case, a count must be stored for each colour sequence, giving space complexity of $\mathcal{O}(|C|^n)$.

C. We next compute the probability distribution, Equation 3.3. We must know $|S_n(G)|$, and for each colour sequence count from the previous step, $|\chi^{-1}(s)|$, to work out the ratio of paths with each sequence to the total number of paths. We can bound this as we have done with the previous step at one division per sequence, and store a floating-point number for each, giving both space and time complexity of $\mathcal{O}(|C|^n)$.

D. Finally, we calculate the entropy as in Equation 3.4. This entails iterating over each element, p_i, in the distribution and calculating $p_i \log(p_i)$, while keeping a running sum. This clearly has $\mathcal{O}(1)$ additional space complexity and the number of arithmetic operations is linear in the number of elements in the distribution, so time complexity is $\mathcal{O}(|C|^n)$.

In summary, the time complexity of calculating E_n is bounded by,

$$\begin{array}{ll} \mathcal{O}(n|V|\langle d\rangle^n + |C|^n) & \text{Average case} \\ \mathcal{O}(n|V|^{n+1} + |C|^n) & \text{Worst case} \end{array} \tag{3.5}$$

and the space complexity by,

$$\mathcal{O}(|V| + |C|^n) \tag{3.6}$$

Additionally we can verify empirically that the running time for the above procedure for calculating E_n increases comparably to an exponential function of n, as shown in Figure 3.2. As we see below, in practice it is unnecessary to calculate E_n directly for $n > 1$ so the exponential running time is not a serious handicap.

3.5.1 Linearity of Path Entropy

As discussed below in Section 3.10, we find an empirical result that, for the graphs and colourings under consideration here, that the path entropy E_n is linear in n. That is,

$$E_n = (E_1 - E_0)n + E_0 \qquad n > 0 \tag{3.7}$$

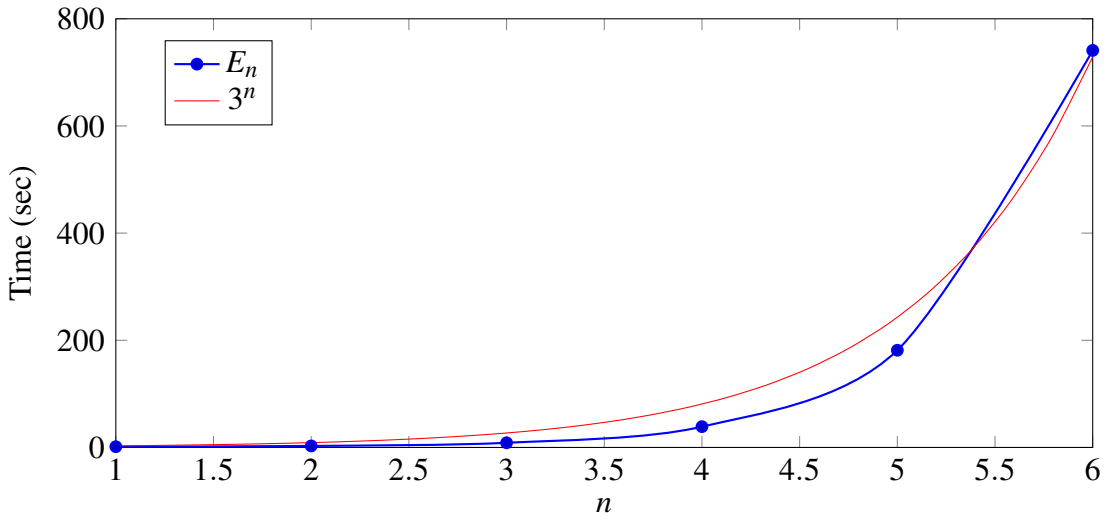


Figure 3.2: Empirical running time of calculation of E_n of the graph of Figure 3.1d for increasing values of n with an implementation in Python running on a 2.4GHz Intel Xeon E5645 CPU.

This observation is significant because as shown by Equation 3.5, the computational work to calculate E_n directly grows exponentially with n. Since it can be worked out simply from E_0, E_1 and n, there is little benefit in the direct calculation.

It is important to note that this result does not hold in general. An easy way to find a counterexample is to construct a graph where the colour of the $(n+1)$th vertex in a path depends not only on the nth but also on previous vertices. Fortunately the paths in the coloured planar graphs that we consider here do not appear to have this property. An interesting theoretical problem that we do not treat here is to precisely determine for which underlying coloured graphs this linear relation holds, and for graphs where it does not, what can be deduced about the path entropy for paths of lengths greater than two.

3.6 Relative Entropy

For completeness, and because it will be used later, we review the concept of *relative entropy* between two probability distributions. This is known in a more general setting as the Kullback-Leibler divergence (Kullback et al., 1951) and is written,

$$D(p\,|\,q) = \sum_i p_i \log\left(\frac{p_i}{q_i}\right) \tag{3.8}$$

for two distributions, $p = \{p_i\}$ and $q = \{q_i\}$. For this to be well-defined, it is required that $p_i = 0$ if $q_i = 0$. Intuitively it gives a notion of distance between two distributions, however this intuition should be taken with a grain of salt: as formulated, in general it will violate the triangle inequality.

In the present context, we consider the distance from a reference graph containing paths R to a given graph G. The reference graph could be the initial conditions for a simulation or experiment or it could be an exemplar or "typical" pattern. This distance in this setting is simply,

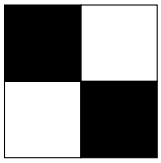
$$D_1(G|R) = \sum_{s \in C^2} p_1(G)(s) \log \left(\frac{p_1(G)(s)}{p_1(R)(s)} \right) \tag{3.9}$$

3.7 Examples in Two Colours

To see how path entropy works in practice, and before considering real examples, let us consider a few simple cases. We first consider very simple patterns in two colours for which entropies can be calculated by hand on rectangular lattices, and then more complex but nevertheless artificial patterns in three colours on hexagonal lattices shown in Figure 3.5.

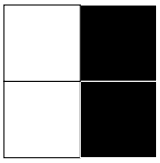
Starting with the simplest possible regular, symmetric two-colour, diagram will illustrate how the measure E_1 captures clustering. In what follows we do not impose periodic boundary conditions, although it would be perfectly natural to do so. Instead, we opt to consider, for clarity of presentation, the graphs exactly as they appear on the page.

Consider a 2x2 checkerboard,



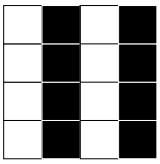
$$|S_1| = 8$$
$$|\chi_1^{-1}(wb)| = |\chi_1^{-1}(bw)| = 4$$
$$E_1 = 1$$

This can obviously be extended to checkerboards of arbitrary size. Furthermore, larger checkerboards will, provided symmetry is preserved, give numerically the same value for E_1 because there are no like-colour adjacencies and for every unlike-colour adjacency is reflexive.



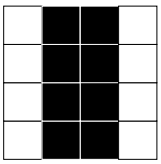
$$|S_1| = 8$$
$$|\chi_1^{-1}(ww)| = |\chi_1^{-1}(bb)| = 2$$
$$|\chi_1^{-1}(wb)| = |\chi_1^{-1}(bw)| = 2$$
$$E_1 = 2$$

Rearranging the squares into stripes, we can see the measure E_1 distinguish between different kinds of regularity. With a little more work, we can see that this value for E_1 is characteristic of stripes one cell wide on a rectangular lattice,



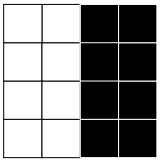
$$|S_1| = 48$$
$$|\chi_1^{-1}(ww)| = |\chi_1^{-1}(bb)| = 12$$
$$|\chi_1^{-1}(wb)| = |\chi_1^{-1}(bw)| = 12$$
$$E_1 = 2$$

Rearranging the stripes into a thick and two thin, however, we see that the E_1 measure counts it as, in some sense, more regular. Or, more to the point, *more clustered*,



$$|\chi_1^{-1}(ww)| = 12$$
$$|\chi_1^{-1}(bb)| = 20$$
$$|\chi_1^{-1}(wb)| = |\chi_1^{-1}(bw)| = 8$$
$$E_1 = 1.89$$

Finally, two thick stripes,



$$|\chi_1^{-1}(ww)| = 20$$
$$|\chi_1^{-1}(bb)| = 20$$
$$|\chi_1^{-1}(wb)| = |\chi_1^{-1}(bw)| = 4$$
$$E_1 = 1.65$$

and this is maximally clustered and a local minimum of the E_1 entropy. It is a local minimum because any change must increase the number of heterotypic edges, and decrease the homotypic ones. Such a change to the distribution of paths can only increase the corresponding entropy.

These minima are interesting. In general, for the two-colour case, the entropy will have two minima: one for a maximally clustered pattern and one for maximally dispersed, checkerboard pattern. The latter is easily seen to be a global minimum as all adjacencies are of the same, heterotypic, type. For the clustered case, while as many edges as possible are homotypic, there still must be an interface between clusters of different colours so not all adjacencies can be the same. The outcome of choosing an arbitrary adjacency at random cannot then be certain, so the entropy must be greater than for the checkerboard.

3.8 Two-Species Epithelia

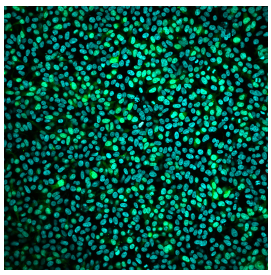
We are now in a position to apply these measures to some real-world cases. We start with some data from the same series as the phase separation study previously mentioned (Cachat et al., 2016). In that study, cells are genetically engineered to vary their level of production of cadherin molecules in response to external regulation using tetracycline. The cadherin molecules govern the adhesiveness of the cells to their neighbours. Two varieties of these cells, differing only in the degree of sensitivity to tetracycline, were mixed randomly together in a 50:50 mixture and allowed to settle. Cell cultures from experiments with, and without tetracycline are shown in Figures 3.3a and 3.3b.

Some processing is needed to take this data into a form where the measures that we define here can be applied. The procedure is relatively straightforward. First, positions and kinds of the nuclei are identified directly from the image. These provide the vertices for our graph. Next, neighbour relationships are derived from the Voronoi tessellation of these points. The results of this procedure on the confocal images are shown in Figures 3.3c and 3.3d.

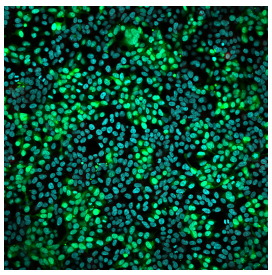
The simulation method that we use to compare to this experimental data is similar to that of Osborne et al. (Osborne et al., 2017) using the Chaste software package (Mirams et al., 2013) and Farhadifar's potential (Farhadifar et al., 2007). In brief, the tissue is described by a potential,

$$U = \sum_{i \in V} \frac{K}{2}(A_i - A_0)^2 + \sum_{i,j \in V^2} \lambda_{ij} E_{ij} + \sum_{i \in V} \frac{\Gamma}{2} P_i^2 \tag{3.10}$$

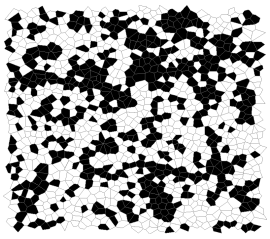
where A_i and P_i are the area and perimeter of the ith cell, respectively, A_0 is the preferred area of a cell (assumed to be uniform in the population of interest in the present scenario), and E_{ij} is the length of an edge between cells i and j (defined to be zero if the cells are not adjacent). The first term represents compression or dilation of the cell away from its



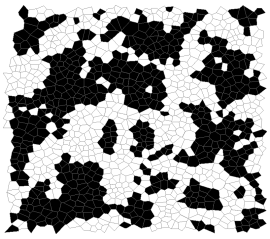
(a) Confocal no tetracycline



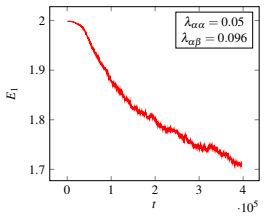
(b) Confocal tetracycline



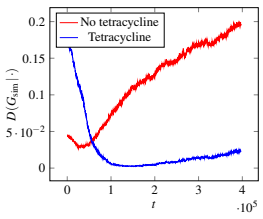
(c) Voronoi no tetracycline



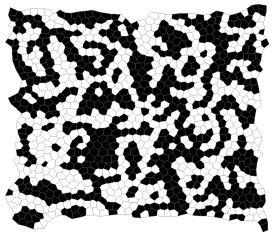
(d) Voronoi tetracycline



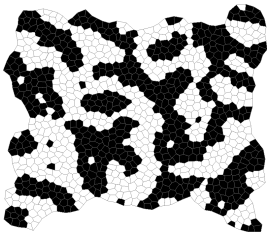
(e) Simulation entropy



(f) Relative entropy



(g) Simulation ≈ no tetracycline



(h) Simulation ≈ tetracycline

Figure 3.3: Top row experimental data, bottom row simulation data. Figures 3.3a and 3.3b show raw confocal images from Cachat et al.'s study of phase separation due to differential adhesion, after 24 hours. All cell nuclei are stained to appear blue, while only the nuclei of the E-cadherin variety appear green. Figures 3.3c and 3.3d show the graph derived from the voronoi tesselation of the cell centroids from the confocal images. Figure 3.3e shows the entropy trace of a typical simulation where heterotypic edges between cells are more costly than homotypic edges. Figure 3.3f shows the data from the same simulation compared using relative entropy with the Voronoi tessellations of Figures 3.3c and 3.3d. Finally, Figures 3.3g and 3.3h show simulated tissues as at the minimum of the relative entropy curves in Figure 3.3f, that is, those that correspond most closely to experiment by our measure. Both curves increase as the simulation becomes yet more clustered than the experiment.

preferred area and the last, the contractility of the perimeter. The constants, K and Γ that trade off the relative importance of these effects are held fixed.

The entire coding for differential adhesion takes place in the middle term of Equation 3.10. λ_{ij} is cost per unit length of an edge between the two cells. For the two-species case, this matrix has entries that are either zero for cells that are not adjacent, or values that depend on the kind of each cell. Heterotypic edges have one value and homotypic another. In what follows, we abuse the notation slightly and interpret $\lambda_{\alpha\beta}$ to mean the cost per unit length of an edge between cells of type α and β, and we use Λ to refer to the matrix of these costs for different cell types.

The simulation proceeds from randomly coloured cells on a regular hexagonal lattice, and the tissue is allowed to relax, in a direction that minimises the potential, rearranging according to the standard topological transitions for foams (C. S. Smith, 1952; Weaire et al., 2009). To avoid settling to a local minimum, at each step vertices are subject to some noise, an additional small force in a random direction.

We model the effect of tetracycline indirectly, representing the induced adhesion effect as the cost of edges. For these simulations we used values for heterotypic edges approximately twice as costly as for homotypic[2] and the result is a time-series of tissue exemplars beginning with cells randomly distributed and gradually developing more structure, or clustering. The claim (Osborne et al., 2017) is that this sequence is representative of the process that occurs *in vitro*. Figure 3.3e shows how the absolute entropy, E_1, of these tissue exemplars changes over time. Clearly it is decreasing overall.

Finally, we can use Equation 3.9 to work out the extent to which tissue snapshots from the numerical simulation are similar to the experimental data. The relative entropies of the simulation to each of the experimental cases, with and without tetracycline are shown in Figure 3.3f. Each has a minimum, and the minimum for the case with tetracycline occurs later, at a stage where the simulation has become more ordered (lower absolute entropy) than without. The tissue exemplars corresponding to these two minima are shown in Figure 3.3g and 3.3h, and they correspond qualitatively well with their experimental counterparts. Our measure makes this impression quantitative.

Notice that the distance to the reference snapshot increases as the simulation progresses. This means that the simulated tissue, for these parameter values, becomes *more* ordered according to our measure than the experimental data. Given a suitably large library of simulation data with which to compare to experiment, one would naturally wish to find one where the distance measure converges to zero in order to make a

[2] In particular, $\lambda_{pp} = \lambda_{ee} = 0.05$ and $\lambda_{pe} = \lambda_{ep} = 0.096$, and in all cases $\Gamma = 0.04$ and $K = 1$

well-supported claim that the simulation parameters are a good fit to the experiment.

3.9 Rate of Pattern Formation

If a time-series of experimental data is available (unfortunately in this instance it is not) it is also possible to compare the rate of pattern formation. We can, however, show how path entropy can be used to quantify the rate of pattern formation with a set of numerical experiments. In these experiments we aim to understand more precisely how differential adhesion affects pattern formation. The salient model parameters are the homotypic edge cost, $\lambda_{\alpha\alpha}$, which is held fixed, an the heterotypic edge cost, $\lambda_{\alpha\beta}$ which we allow to vary. Other parameters such as perimeter contractility, Γ, the area pressure constant, K, and the amount of noise, Z, we also hold fixed. The results are shown by plotting E_1 for various values of $\lambda_{\alpha\alpha}$ in Figure 3.4.

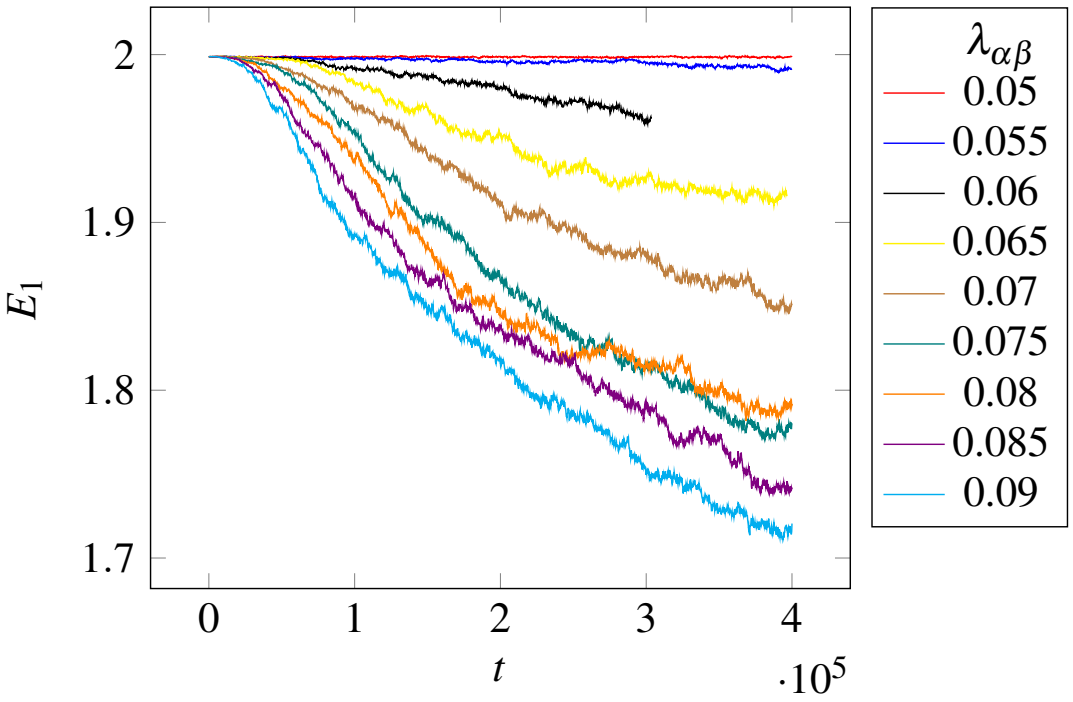


Figure 3.4: Path entropy time series for simulations with various values for the heterotypic edge cost, $\lambda_{\alpha\beta}$. In all cases the homotypic value is $\lambda_{\alpha\alpha} = 0.05$.

As suggested by the simple examples in Section 3.7 and can be seen in Figure 3.3, lower entropy values correspond to more "dappling" patterns as they were described by Turing. The physical reason for the emergence of the pattern, in discrete models such as the Vertex or Cellular Potts models governed by a potential such as Equation 3.10 is quite simple. The heterotypic perimeter of a patch is expensive compared to the

homotypic interior, so the dynamics simply arise from the process of minimising (up to the appropriate constants) the ratio of perimeter to surface area. Absent topological constraints, the shape that accomplishes this minimisation is a circle. In an equal mixture of cells constrained to be a planar graph, both kinds of cells cannot form circular patches simply because it is not possible to tile a plane with circles. Therefore competing but symmetric tendencies of each kind of cell to try to form circular patches results in the familiar pattern.

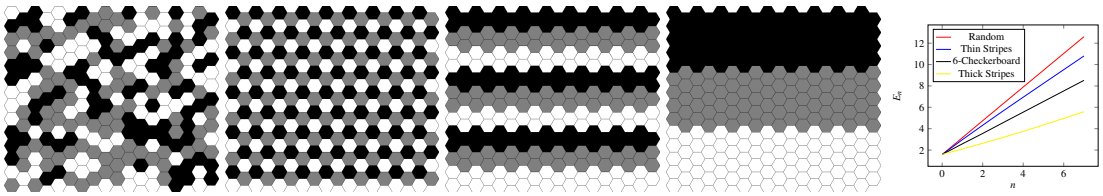
Given this understanding of the process, what we can read from the figure is, all else being equal, the greater the difference between the homotypic and heterotypic edge costs, the more rapidly the entropy of the tissue decreases. It takes about twice as long for the simulation with a heterotypic edge cost, $\lambda_{\alpha\beta} = 0.07$ as does the one for which $\lambda_{\alpha\beta} = 0.09$ to reach degree of pattern present that corresponds to $E_1 = 1.9$. When the heterotypic cost is only slightly larger than the homotypic cost, it may take much longer indeed to achieve that same degree of patterning.

Not shown are cases where the homotypic cost is allowed to vary, but the conclusions are straightforward and readily apparent from the time-series of our E_1 measure for them. Larger values of $\lambda_{\alpha\alpha}$ that are still smaller than $\lambda_{\alpha\beta}$ do result in patterning, but more slowly. This makes sense because these larger values are more rigid and as a result the entire system changes more slowly and the topological transitions that are necessary for pattern development due to cell migration less frequent. When $\lambda_{\alpha\alpha}$ is allowed to be greater than $\lambda_{\alpha\beta}$, the resulting pattern is very different because now rather than minimising the number of heterotypic edges they should be maximised. In this way we get patterns much like a checkerboard as predicted in Section 3.7. While these underlying mechanisms are well known, their effect is clearly exposed by studying the behaviour of E_1.

3.10 Examples in Three Colours

The patterns in Figure 3.5 are all regular, except for the first, which is random. The random pattern is in fact representative of the initial conditions of the simulations which we will see later. They use three colours and a regular hexagonal lattice. This has some important consequences for the minimal entropy in a three-colour setting as we will see.

Figure 3.5 shows some example graphs in three colours and the corresponding path entropies. As usual, the number of cells of each colour is equal. Again, we include



(a) Random (b) 6-checkers (c) Thin stripes (d) Thick stripes (e) Path entropies

Figure 3.5: A selection of three-coloured planar graphs Figure 3.5e shows the path entropies, E_n, for these graphs, for path lengths n from 0 to 7.

a randomly coloured graph and we include the generalisation of a checkerboard to a hexagonal lattice. We also include thin and thick stripes.

Some observations about the minima of the entropy can be made here and they are different from the two colour case. The example with thick stripes, or greater clustering, has lower entropy than the others and it is a minimum by the same argument from Section 3.7, namely that any change can only increase the entropy by lessening the number of homogeneous edges.

In the three-colour case, however it has a lower entropy than the maximally dispersed, equivalent of the checkerboard. This is because it is not possible to colour a hexagonal lattice with only two colours while respecting the constraint that no two adjacent cells may have the same colour. Three colours are needed. This means that it is no longer true that for the maximally dispersed case all adjacencies are identical. The hexagonal checkerboard therefore no longer corresponds to a global minimum of E_1. In fact the maximally clustered graph must now be the global minimum.

For these examples, the entropy for longer path lengths was also calculated directly. The results, shown in Figure 3.5e clearly illustrate that there is no benefit to the extra computational cost of calculating path entropy for paths of longer than 2 cells. This provides some further justification to our choice to confine our attention to E_1. The reasoning about the minimum of E_1 for the three-colour case shows that this measure appropriately captures the degree of clustering or homogeneity.

3.11 Three-Species Epithelia

Turning finally to the examples from Figure 3.1, we briefly study the patterning dynamics of epithelia consisting of three cells. We show that the E_1 metric can also be employed to evaluate whether one can distinguish the rate of pattern formation in

systems with multiple cell types. As with the two-cell case, we consider interactions between cell types, but now form a 3x3 matrix,

$$\Lambda = \begin{bmatrix} \lambda_{rr} & \lambda_{rw} & \lambda_{rb} \\ \lambda_{wr} & \lambda_{ww} & \lambda_{wb} \\ \lambda_{br} & \lambda_{bw} & \lambda_{bb} \end{bmatrix} \tag{3.11}$$

accordingly as an edge is between red, r, white, w, or blue, b cells. We presume that this matrix is symmetric, and indeed it can always be symmetrised without changing the behaviour simply by taking, $\lambda'_{\alpha\beta} = \lambda'_{\beta\alpha} = \frac{1}{2}(\lambda_{\alpha\beta} + \lambda_{\beta\alpha})$.

We consider four cases, in an attempt to find a regime where the presence of a third kind of cell materially affects phase separation and pattern development. Namely,

I. Homotypic red and blue edges inexpensive, homotypic white edges are very expensive. Heterotypic edges with a white cell are very inexpensive and heterotypic red-blue edges are relatively expensive. Absent white cells, this behaves like the typical red-blue dappled pattern. Adding white cells should have them maximally dispersed.

II. As with Case I, but the relationships to white cells inverted. Homotypic edges among white cells are now very inexpensive, and heterotypic ones are now very expensive. This is expected to form round patches of white cells.

III. All homotypic edges have the same, low cost. Heterotypic edges with white cells are relatively inexpensive and red-blue edges are relatively expensive. The low cost of white-heterotypic edges produces long, thin, white borders between red and blue regions.

IV. As with Case III, but with the heterotypic costs inverted. Edges between a red or blue and a white cell are now expensive and red-blue heterotypic edges are relatively inexpensive. This produces results very similar to Case II.

For these numerical experiments, in each case, the proportion of white cells was varied from 0 to 33%. The results of calculating time-series for E_1 are shown in Figure 3.6.

This cursory search of four regions of the parameter space does not uncover a regime where a third kind of cell affects the rate or degree of pattern formation. This fact is made quite clear by the E_1 measure, whose rate of change is essentially the same for all of the cases. It remains an open area of research whether or not there is a regime where

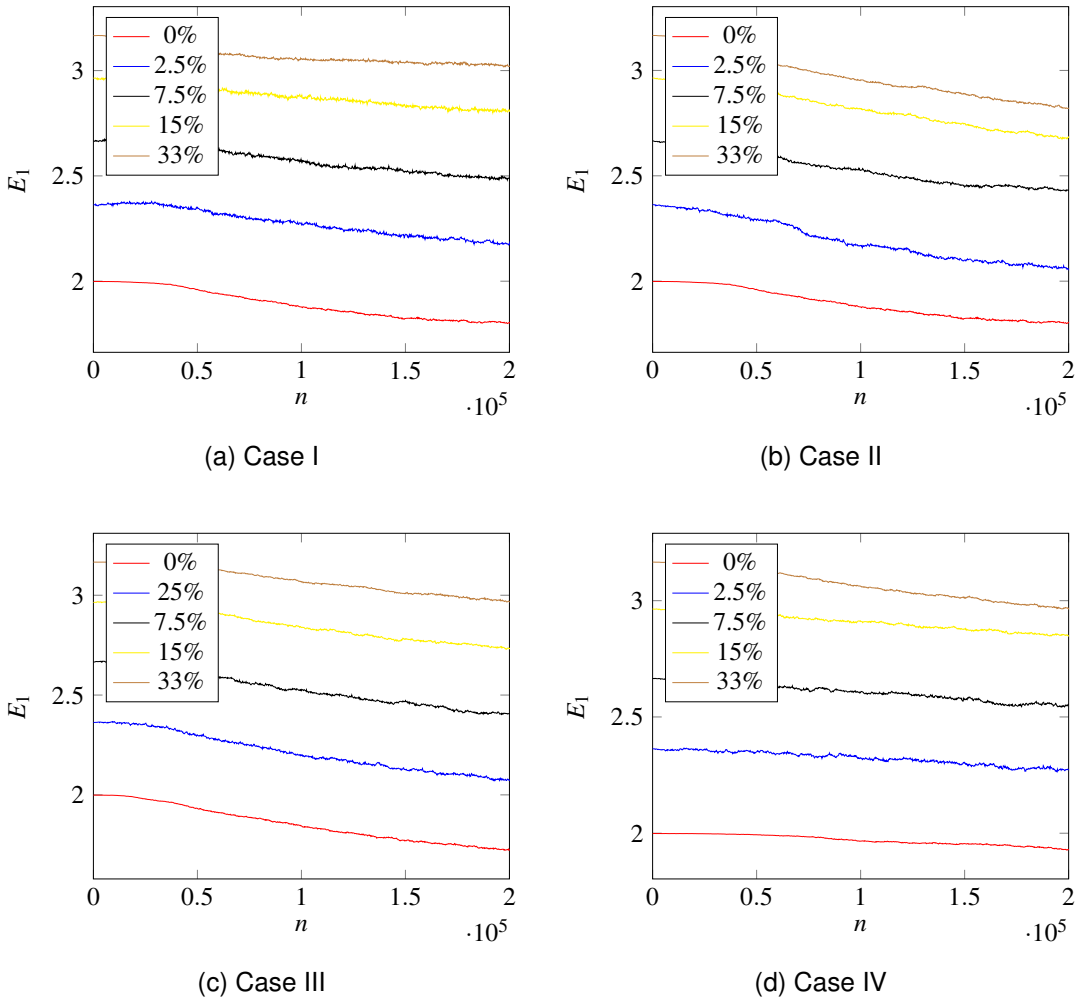


Figure 3.6: Entropy for various population fractions of white cells, for each of the cases above. The salient observation is that the slope of the curve, the rate at which entropy changes, do not vary perceptibly with the amount of white cells.

the presence of a third kind of cell accelerates or retards pattern formation by acting analogously to a lubricant or a glue.

3.12 Tissue Library for Parameter Fitting

A natural supposition, given this ability to measure how well patterns in simulated tissue graphs correspond to experimentally derived ones, is that we may be able to estimate the parameters in the Farhadifar potential, Equation 3.10, to the experimental data. This possibility is suggested by the observation that not only is the degree of patterning measurable using our technique, so is the rate of pattern development. These two measures, E_1 and its time derivative could in principle be used for parameter

estimation. Equally they could be used as predictors of experimental behaviour, for example estimating the concentration of a certain antibiotic required for a given rate of phase separation.

Up to normalisation, the parameters Λ and Γ do correspond to physical phenomena, after all. This kind of fitting is indeed possible, with some limitations. The main limitation is that it is not possible to distinguish, within the region of interest between equally good pairs of parameters, (Λ, Γ), along an iso-surface in the phase diagram (Staple et al., 2010; Magno et al., 2015). However, holding one fixed (Γ) it is indeed possible to derive an estimate of the corresponding value for Λ.

The procedure is simple but would require a large library of simulation data. For each parameter value, the time-series of E_1 can be calculated and stored, along with other statistics of interest (such as the degree distribution of cells). Data emanating from experimental imagery, processed into a coloured graph using the Voronoi tessellation or other techniques, can then be compared, and a best guess at the parameters arrived at. The time derivative of E_1 is important because often different adhesion values (Λ) that produce similar patterns can be distinguished by the rate at which the patterns appear.

Producing such a library is a very computationally intensive task. For the present work, we have simulated only a small subspace of possible parameter choices for two cells, and for three, and without necessarily reaching a steady state in all cases this has consumed several CPU-decades of processing time. Furthermore for accurate distributions, the tissue size should be as large as possible and at present tissues larger than about 5000 cells are prohibitive.

Despite the challenges, it is worthwhile to create and make available such a resource which the authors believe would be a valuable quantitative tool for synthetic morphology research.

3.13 Conclusions

Aside from application in synthetic morphology, the method presented can also be adapted to analyse samples of natural tissues and applied to the study of cancer progression. Recent successes using deep learning neural networks (Esteva et al., 2017) to characterise cancer progression in tissue imagery samples are instructive. In that study accuracy rates with deep learning were comparable to trained pathologists but the technique does not permit inspection or reverse-engineering to identify the salient features being recognised. Successes using similar techniques have also been reported

for identifying certain cardiovascular pathologies (C. Xu et al., 2017). By contrast, our measure has much more stringent requirements on input data — we require input in the form of a coloured graph — but its principle of operation is straightforward to understand.

There is an important limitation when applying this technique to imagery from naturally occurring, as opposed to synthetic, tissue. Path Entropy is defined by cell types and their adjacencies. Synthetically engineered tissue designed to study mechanical interactions among cells is much more regular than its naturally occurring counterpart. This means it is correspondingly easier to extract the information needed to calculate path entropy from images of synthetic tissues. Accommodating structural heterogeneity in naturally occurring tissue likely requires segmentation techniques that consider actual cell boundaries and not a Voronoi tessellation derived from nuclei as we have done here. Advances in microscopy and optical technologies make possible high-throughput analysis and simultaneous measurements of proteins and other molecules (such as miRNA) in histological specimens and tissues micro-arrays which allow the identification of subpopulations of genetically similar cells within tissue samples, using measurement of loci-specific fluorescence *in-situ* Hybridization (FISH) spot signals for each nucleus (Croce, 2008; Guillaud et al., 2010). The use of neural networks to perform segmentation at the tissue level has been shown and remains a current topic of research (Reddick et al., 1997; Wenlu Zhang et al., 2015; Su et al., 2017). These methodologies could facilitate the construction of the graph underlying an epithelial tissue and suggest an appropriate extension of the metric proposed in this work.

In this chapter, we have defined a specialised class of entropy measures, path entropies, on adjacency graphs designed to quantify the degree of patterning present in cellular tissues and noted some of its interesting properties. We have demonstrated how this measure can be used on two dimensional epithelial tissues to establish a correspondence between experimental and simulation data that quantifies the impression of similarity between the patterns expressed. We have further demonstrated how the measure generalises to tissues consisting of three species and noted some differences from the two species case. Finally, we have proposed, for the specific application of synthetic morphology, the establishment of a library of tissue data upon which these measures can be calculated, to assist in parameter estimation, providing a useful quantitative tool for synthetic morphology.

Chapter 4

The Structure of a Mouse

Abstract. We apply an information-theoretic measure to anatomical models of the Edinburgh Mouse Atlas Project. Our goal is to quantify the anatomical complexity of the embryo and to understand how this quantity changes as the organism develops through time. Our measure, *Structural Entropy*, takes into account the geometrical character of the intermingling of tissue types in the embryo. It does this by a mathematical process that effectively imagines a point-like explorer that starts at an arbitrary place in the 3D structure of the embryo and takes a random path through the embryo, recording the sequence of tissues through which it passes. Consideration of a large number of such paths yields a probability distribution of paths making connections between specific tissue types, and *Structural Entropy* is calculated from this (mathematical details are given in the main text). We find that Structural Entropy generally decreases (order increases) almost linearly throughout developmental time (4d-18d). There is one 'blip' of increased Structural Entropy across days 7-8: this corresponds to gastrulation. Our results highlight the potential for mathematical techniques to provide insight into the development of anatomical structure, and also the need for further sources of accurate 3D anatomical datato support analyses of this kind.

The work presented in Chapter 4 is reprinted with permission from William Waites and Jamie A. Davies, "Emergence of Structure in Mouse Embryos: Structural Entropy Morphometry Applied to Segmented Anatomical Models", *Journal of Anatomy* (2019). The work was conceived by both authors. I developed the mathematical formulation of *Structural Entropy* by extending *Path Entropy* to the continuous setting, implemented the software for processing the segmented anatomical data, conducted the analysis and drafted the initial text; both authors contributed equally to the final text.

4.1 Introduction

In his important text on developmental biology (Kauffman, 1993), Kauffman argues that order in living creatures arises from a combination of evolution and self-organisation. A remarkable fact about this beautiful text is that the meaning of its title, "Origins of Order" is left essentially implicit: the meaning of the word "order" is never defined. It is discussed extensively, contrasted with "chaos", and asserted as a property of various remarkable observations about fitness landscapes, but we may continue to wonder what, precisely, is meant by "order". It is not hard to account for this ambiguity; exactly what should be meant by order, or related words such as structure or complexity as they apply to biological organisms, is not at all obvious. Indeed, authors such as Grizzi et al. (2005) consider the meaning of anatomical structure in detail, making the key point that "complexity can reside in the *structure* of the system," and suggest the use of mathematics to quantify this, without explaining precisely how.

In this paper, we offer a possibility for quantifying a particular kind of order: the physical structure that develops as an organism grows. We call this measure *Structural Entropy*.

Structural Entropy is a quantity calculated on an abstract representation of the organism's anatomy. To understand how Structural Entropy works, it is helpful to consider the general concept of entropy in Information Theory (Shannon, 2001). The quantity now known in that field as entropy was originally called "uncertainty" by Shannon. Given a probability distribution over some set, if the set is dominated by many equally likely elements (as in a normal pack of cards), the outcome of choosing one at random is very unpredictable, and hence the entropy will be large. If some elements are much more likely to be chosen than others (eg in a pack of cards containing 50 jokers and 2 aces of spades), we can be a bit more certain about the outcome and the entropy will be smaller. In this report, we use this concept, as well as our previous work (Waites; Cavaliere, et al., 2018) to construct such a probability distribution using the topology of an anatomical model, augmented with geometrical data. This distribution says how likely it is to find a notional particle, allowed to travel freely through the embryo, in any given embryonic tissue.

There is, however, a major obstacle in applying this measure, especially in developmental anatomy: the lack of sufficient good quality data to support applications of the type we propose. What is required is a complete library of accurate digital models ("atlases") of embryonic anatomy, from closely-spaced stages of development, each

digitally annotated so that each pixel (2D) or voxel (3D) is labelled with the identity of the tissue in which it lies. We will refer to this labelling process as "tagging". The best current approximation of such a data library is the Edinburgh Mouse Atlas or eMouseAtlas (Davidson et al., 2001; R. A. Baldock et al., 2003; J. H. Christiansen et al., 2006; Richardson; Venkataraman; Stevenson; Yang; Nicholas Burton, et al., 2009; Richardson; Venkataraman; Stevenson; Yang; Moss, et al., 2013; Armit; Venkataraman, et al., 2012; Armit; Richardson, et al., 2015). The eMouseAtlas was constructed by digitization of serial sections of complete mouse embryos at closely-spaced stages of development. The different tissues in each digital image were identified and delineated by expert embryologists, who tagged the different regions of the embryos with the tissue identity. These tagged images were then assembled into 3-dimensional models of the corresponding embryo, and the datasets are available online. We use the eMouseAtlas to illustrate how Structural Entropy can be calculated and show that it captures structure increasing with time. However, there are very few datasets of this kind available.

The eMouseAtlas contains 3D tagged anatomical models of house mouse (*Mus musculus*) embryos at a selection of pre-natal stages of development. It is the best freely available dataset of its kind for demonstrating the kind of analysis that we suggest. Nevertheless, it has some defects and inconsistencies which we detail in Section 4.3.1. More broadly, good quality 3D tagged anatomical models for every developmental stage are simply not available for any organism. The similarly named Worm Atlas (Altun et al., 2002-2018), which uses the model organism *Caenorhabditis elegans*, contains a wealth of resources: diagrams of adult organisms, cell lineages and gene expression data, but only scattered anatomical models. There is a wealth of magnetic resonance image data available for the human brain (Van Essen et al., 2013), but this is intentionally distributed in a minimally processed way in order to encourage development of techniques for identifying structures within images and further processing. These data are therefore not immediately amenable to the analysis that we advocate here, though it is possible to imagine intermediate processing of those images that could make it so.

Much previous work on the complexity of models of anatomical features is from neuroscience. Several authors characterise complexity as a dynamic quantity. Tononi et al. (1994) introduced an information-theoretic measure called *Neural Complexity (NC)*. They measure the temporal patterns of signals through neural networks and claim (also Sporns et al. (2000)) that these patterns must depend strongly on the underlying anatomical structure. Later authors such as Fan et al. (2017) consider information theoretic measures on the neural connectome directly. Horn et al. (2014)

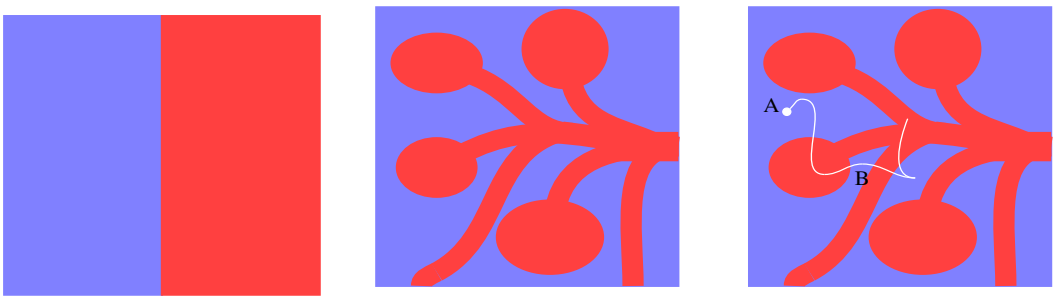
use a random-walk approach at a much finer grain to find agreement between the structural and functional connectivity for the brain's default-mode network. Chan et al. (2014) use this technique to measure desegregation of brain networks with age and long-term memory function. For practical reasons, suitable data pertaining to human developmental anatomy is difficult to obtain (Huang et al., 2006; Mietchen et al., 2009), particularly for early developmental stages; therefore computational morphometry is applied mainly to the study of diseases related to ageing (Testa et al., 2004; Matsuda, 2013). We believe that our Structural Entropy measure might also provide a useful diagnostic signal in the context of this kind of ageing study and suggest this as an area of future research.

One of us (Jamie A Davies, 2016) considered a similar question to that which concerns us here, using a different subset of the eMouseAtlas data. Davies considered the text annotations, and the number of terms required to describe each developmental stage, arguing that the greater number of terms needed, the greater the complexity. From these data, Davies showed that the number of vocabulary terms increases exponentially over time. We show here that what Davies' result provides is, in fact, a lower bound on order.

In this chapter we confine ourselves to developing Structural Entropy in the context of the data from the eMouseAtlas and show that it captures something of the intuitive idea of increasing anatomical order as development progresses. This line of reasoning relies on the assumption that the anatomical analysis is a faithful representation of the underlying structure in the organism. We show that Structural Entropy appears reasonably robust to inconsistencies in manual analysis and tagging.

4.2 Methods

Because the detailed mathematical description of our methods (Section 4.2.2) may not be easily accessible to all readers, we provide an additional illustrated description written in non-technical English. This account (Section 4.2.1) captures the essence of how our analysis works but necessarily involves informal and imprecise analogies; readers wishing to criticise, replicate or build on our work are strongly advised to engage directly with Section 4.2.2.



(a) Two tissues, each occupying half of the embryo, in a simple spatial arrangement.

(b) Two tissues, each occupying half of the embryo, in a rich spatial arrangement.

(c) A random path (B) from a random starting point (A) through and between the tissues.

Figure 4.1: Tissues in simple and rich spatial arrangements and an example random path.

4.2.1 Informal Description of Method

As mentioned in the introduction, our concept of Structural Entropy is related to Claude Shannon's concept of "uncertainty" (later called "entropy") in the field of Information Theory. This is a measure of disorder, or unpredictability, in a set of data. If the outcome of a random dip into a bag of data elements is known with high probability (eg if 90% of the numbers in the data set were '1'), then the predictability would be high and the entropy low. If the outcome of the random dip were only known with very low probability (eg the numbers in the data set were truly random), then the predictability would be low and the entropy high.

The structure of an embryo, or any other biological object, can be modelled as a bag of data, each data element comprising 3D coordinates, (x, y, z), that specify its position and a tag that specifies the tissue name at that point. A naive approach to measuring the degree of order might therefore be to make many random dips into the data set for an embryo, and calculate the probability distribution of finding a tag for different tissues (eg "ectoderm", "mesoderm" etc. for a gastrulation-stage embryo), and use this to make a measure of structure. This approach, however, has a serious problem: an embryo that consisted of two tissues each of which occupied one half of the embryo (Fig 4.1a) would have the same probability distribution as one that consisted of the same 50/50 mix of two tissues in a rich spatial arrangement (Fig 4.1b). Clearly, a measurement that would ignore such rich anatomical organisation would not be useful.

To avoid this problem, we consider not simply random dips into embryological data,

but random paths taken through the embryo. We begin at a random point and allow a particle to traverse a random path (Fig 4.1b). Then, after doing this for many starting points and paths, we can calculate the probability distribution that a path starting in tissue 1 (say, ectoderm) finishes in tissue 2 (say, endoderm) within a certain number of steps. It can be seen intuitively that the probability distributions that would result from the anatomy in Fig 1a, where most short paths would never leave their starting tissue, would be very different from those resulting from the anatomy in Fig 4.1c. This way of proceeding does, therefore, capture a measure of anatomical richness as well as simple proportions of composition.

We use these path-based probability distributions to calculate Structural Entropy, as defined in Section 4.2.2.2. This involves one important adjustment. Clearly, the more different tissues there are in an embryo, the more alternatives there are for the tissue-type tag corresponding to a spatial position, and the higher the maximum entropy. To avoid our measure being dominated by this trivial effect, we calculate the maximum possible entropy (highest possible disorder) of each embryonic stage by imagining all its tissues being present in an arbitrarily fine, random jumble. We then divide our measure of Structural Entropy from that embryo by the maximum possible entropy, to provide a normalised measure of Structural Entropy that can be compared, fairly, between different embryonic stages that contain different numbers of tissues.

4.2.2 Technical Description of Method

4.2.2.1 Path Entropy

We previously defined Path Entropy as a measure of patterning on tagged graphs (Waites; Cavaliere, et al., 2018) and we give a brief summary here. See Section 4.5 glossary for the meaning of “graph” and “tagged” in this context. In our original treatment (Waites; Cavaliere, et al., 2018) we used the word “colour” instead of tag, as is usual in computer science.

The intuition underlying Path Entropy is as follows. The standard notion of entropy for two-dimensional images is constructed from the probability distribution of pixel colour values (Tsai et al., 2008; Gonzalez et al., 2004; Mangin, 2000). The probability of a pixel being green, say, is just the fraction of pixels that are green. In order to capture more structure, we generalised it in two ways. First, rather than a regular rectangular lattice as in a digital image, we allow an arbitrary graph, with each vertex having a tag. Second, we consider not only the probability of a vertex having a given tag, but

the conditional probability distribution of its neighbours' tags. This is then extended to neighbours' neighbours and so forth, for paths of a given length.

More formally, let $G = (V, E, C, \chi)$ be a tagged graph, where V and E are vertices and edges (see glossary, Section 4.5), C is a set of tags, and χ is a function that gives the tag corresponding to a vertex. In other words if v is a vertex in this graph, then $\chi(v)$ is its colour. This is enough to re-create the standard image entropy mentioned above by counting the number of vertices with tag a and dividing by the total number of vertices,

$$p(a) = \frac{|v \in V, \chi(v) = a|}{|V|} \tag{4.1}$$

After all, a pixel grid can be thought of as a graph where each pixel is a vertex and pixels are adjacent if they share an edge.

Instead of considering the vertices on their own, consider now how they are connected together. A path in the graph is a sequence of vertices connected by edges (loops are allowed). A path of length n is a sequence of $n+1$ vertices connected by n edges in the graph. Define the function χ_n to be the analogue of χ: rather than giving the tag for a single vertex, $\chi_n(\sigma)$ gives the sequence of tags corresponding to a sequence of vertices σ. If we call S_n the set of all paths of length n in the graph, then we can find the probability of a tag sequences s by analogously counting all of the paths that have that sequence,

$$p_n(s) = \frac{|\{\sigma \in S_n, \chi_n(\sigma) = s\}|}{|S_n|} \tag{4.2}$$

The nth order Path Entropy is then defined simply the entropy of this distribution,

$$E_n = -\sum_{s \in C^{n+1}} p_n(s) \log\left(p_n(s)\right) \tag{4.3}$$

4.2.2.2 Structural Entropy

A 3D anatomical model is not an abstract graph with edges and indistinguishable vertices. It consists of regions in space that have particular shapes, each region has a certain tag, and regions can be adjacent to each other. To extend Path Entropy to a setting where it can be applied to regions with spatial extent, accounting for their geometrical structure, we reason as follows.

Begin with a space, X, with a Lebesque measure. In two or three dimensions, this corresponds to normal Euclidean space, but for generality we are not concerned so long as length, area, volume and any higher-dimensional analogous concepts are well-defined and can be summed or integrated over. Let this space be sub-divided in to a set

of regions, $R = \{R_i\}$, and ask what the probability is, if a point is chosen uniformly at random, that it will be found in a given region, R_i. This probability, is the fraction of the total volume occupied by that region,

$$p(R_i) = \frac{\int_{R_i} dx}{\sum_j \int_{R_j} dx} \tag{4.4}$$

Analogously to the discrete case of image entropy, define the function χ to yield the tag for a given region. We can find the probability of a certain tag, c, by adding up the probabilities of choosing a point in a region with that tag,

$$p_c = \sum_{\{R_i \in R, \chi(R_i)=c\}} p(R_i) \tag{4.5}$$

We would like to extend this in a way that accounts for the shape of the regions and their adjacencies with each other. To provide some intuition to guide us, we use the idea that structure is related to communication. In a living organism, the shapes that different anatomical systems have are strongly influenced by communication. Nutrients and chemical signals travel along physical pathways and diffuse across boundaries. The travel of these molecules from one system to another (possibly undergoing transformation along the way) is a kind of communication. Exchange of molecules between systems is facilitated by relatively larger shared boundaries. This constraint influences the shape of the system. Minimising boundary size results in a spherical shape so the degree to which diffusion and hence communication is prioritised is the degree to which the volume occupied by the system differs in shape from a sphere.

Proceeding on this basis, imagine that the randomly chosen point somewhere, in some region, is a notional molecule or particle. This particle is allowed to drift randomly in each region. When it comes to the edge of a region, adjacent to another it may diffuse across this boundary. After some time, the particle will be found in some region, possibly having traversed some others. If s_1 represents the path taken through the first region, s_2 the path taken through the second, and so forth, the sequence, $s_1, \ldots, s_n$, represents the trajectory of the particle. There is a tag that corresponds to each region, so there is a tag sequence that corresponds to this trajectory. If we can work out from the data all of the tag sequences that can be produced by the notional wandering particle, then we can ask, as we did before (Equation 4.2) for a probability distribution of tag sequences. We call the entropy of this distribution the *Structural Entropy*.

One way to work out the distribution of tag sequences is to consider all the possible paths that this particle might take through the various regions from each starting, to each

ending point. This approach affords a large degree of flexibility for modelling: each region can contribute in different ways to the action, encoding more information than is present in the spatial relations themselves. However the data necessary for such an ambitious approach are not available and it is far from clear how to appropriately model the contributions of different anatomical regions to the complexity of the organisms as a whole.

We restrict the question to what can be answered with the available data. To this end, we ask instead, given that the particle started in a region with the tag c_i-, what is the chance that it eventually ends up in one with the tag c_j? This question allows us to quantify the a the notion of communication or interaction mediated by this notional particle between regions of different type, over *any* path. This answer to this question is the basis for our definition of Structural Entropy.

To simplify matters, let us suppose that each R_i has a distinct tag. This can be done without loss of generality because it is always possible to construct such a set. Let,

$$R' = \left\{ \bigcup R_i, \chi(R_i) = c, c \in C \right\} \tag{4.6}$$

where $\bigcup$ denotes spatial union. R' is a set of distinctly tagged regions.

We will model the trajectory of the particle as a Markov process. A Markov process (in discrete time) is characterised by a stochastic matrix, $Q = \left[q_{ij}\right]$. Each element of this matrix, q_{ij}, represents the probability that the notional particle, if it is in a region with the tag c_i, will cross into a region with the tag c_j at the next time-step.

The starting position of the particle is given by Equation 4.5. That is, we assume that the particle has a chance to be starting in region R_i proportionally to its share of the volume. We write this distribution of starting positions as the column vector $\mathbf{p} = [p_i]$. After one time-step, the probability distribution of where the particle will be found is given by $Q\mathbf{p}$. After n time-steps, the distribution is be given by $Q^n\mathbf{p}$. Using this, we can define the nth order Structural Entropy directly analogously to the nth order Path Entropy by,

$$E_n = (Q^n\mathbf{p}) \cdot \log(Q^n\mathbf{p}) \tag{4.7}$$

where the notation $\log(\mathbf{x})$ for some vector $\mathbf{x} = [x_i]$ means $[\log(x_i)]$, and the product $\cdot$ is the standard vector dot- or inner-product.

If the regions, R_i are connected, there is no partition in the graph of their adjacencies, there are no islands, then the Markov process described by Q is ergodic. A particle beginning in any region will eventually visit every other, and there will be a solution to

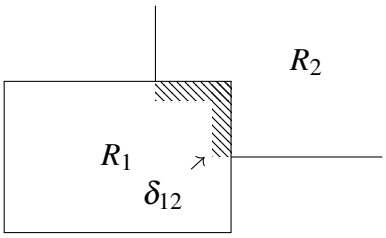
the equation,

$$\pi = Q\pi \tag{4.8}$$

giving the unique stationary distribution π that is independent of the starting position (**p**) (Pinsky et al., 2010). We define the entropy of this distribution, if it exists, to be the Stationary Structural Entropy,

$$E_\pi = \pi \cdot \log(\pi) \tag{4.9}$$

A method to calculate the Q remains to be determined. A principled way would be to say that a particle sufficiently close to a R_i's boundary has a chance of diffusing across the boundary into R_j proportionally to the fraction of R_i's total surface area that is adjacent to R_j. Consider the figure at right, showing the adjacency between R_1 and R_2. The shaded liminal region δ is taken to be the region where diffusion can happen. The liminal region is a buffer around of R_1, extending outwards from the boundary, wherever there is an adjacent region. We can now work out the transition probabilities,



$$q_{11} = \frac{\int_{R_1 - \delta_{12}} dx}{\int_{R_1} dx} \tag{4.10}$$

$$q_{12} = \frac{\int_{\delta_{12}} dx}{\int_{R_1} dx} \tag{4.11}$$

The chance to leave R_1 for R_2 is given by the fraction of R_1's volume that is near enough to R_2 for the particle to diffuse across the boundary. The chance to remain in R_1 is the fraction of its volume that is not sufficiently close to another region.

There are several reasonable ways to define the liminal region, δ. The most natural approach, suggested by the diagram, is for it to be the region within some constant distance of the boundary. This fails on practical grounds – namely that determining the patch of the surface of R_1 that is adjacent to R_2 relies on the underlying data being sufficiently accurate and that there is a portion of their surfaces that are indeed spatially coincident. This is not actually the case in practice with the available data.

We work around this limitation of the data in the following way. We determine the portion of R_1's volume is near to R_2 by dilating the latter by a small amount, k, and take the intersection of R_1 and the dilated region, denoted by $D(R_2, k)$. We then calculate the transition probabilities by first calculating the relative volumes of a region and its

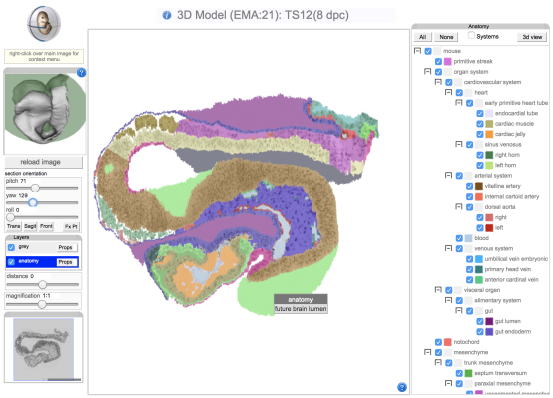


Figure 4.2: The on-line eMouseAtlas viewer inspecting a cross-section of the tagged embryo at Theiler stage 12.

liminal volumes with adjacent neighbours,

$$v_{ij} = \begin{cases} \int_{R_i \cap D(R_j,k)} dx & i \neq j \\ \int_{R_i} dx - \sum_{i \neq j} v_{ij} & \text{otherwise} \end{cases} \tag{4.12}$$

and then construct the transition probabilities by normalising,

$$q_{ij} = \frac{v_{ij}}{\sum_j v_{ij}} \tag{4.13}$$

4.3 Results

4.3.1 The Mouse Atlas

The eMouseAtlas contains, in addition to genomic data and a large amount of structured metadata, 3D geometrical models of the delineated anatomy of mouse embryos at several stages of pre-natal development. In total, there are 69 embryo models available for download (Armit; R. Baldock, et al., 2017) covering Theiler's morphological stages (Theiler, 1989) 7 through 26. Of these, the majority contain untagged 3D reconstructions and Optical Projection Tomography (OPT) images, but there are 22 with anatomy delineations.

Figure 4.3 shows some basic information about the delineated datasets. Each 3D dataset is reconstructed (Hill et al., 2015) from a series of 2D images arranged in layers. The datasets are made available in the Woolz format (Piper et al., 1985) which is both compact and suitable for computation of spatial operations such as union, intersection, convex hulls, and so forth. We will be concerned with volumes of and adjacency

relations between tagged elements, or in other words the sizes of anatomical regions and which are in physical contact with each other. For this reason, in addition to the count of tagged elements in each dataset, Figure 4.3 shows counts of tagged geometrical elements with non-zero volume and those that touch at least one other tagged element.

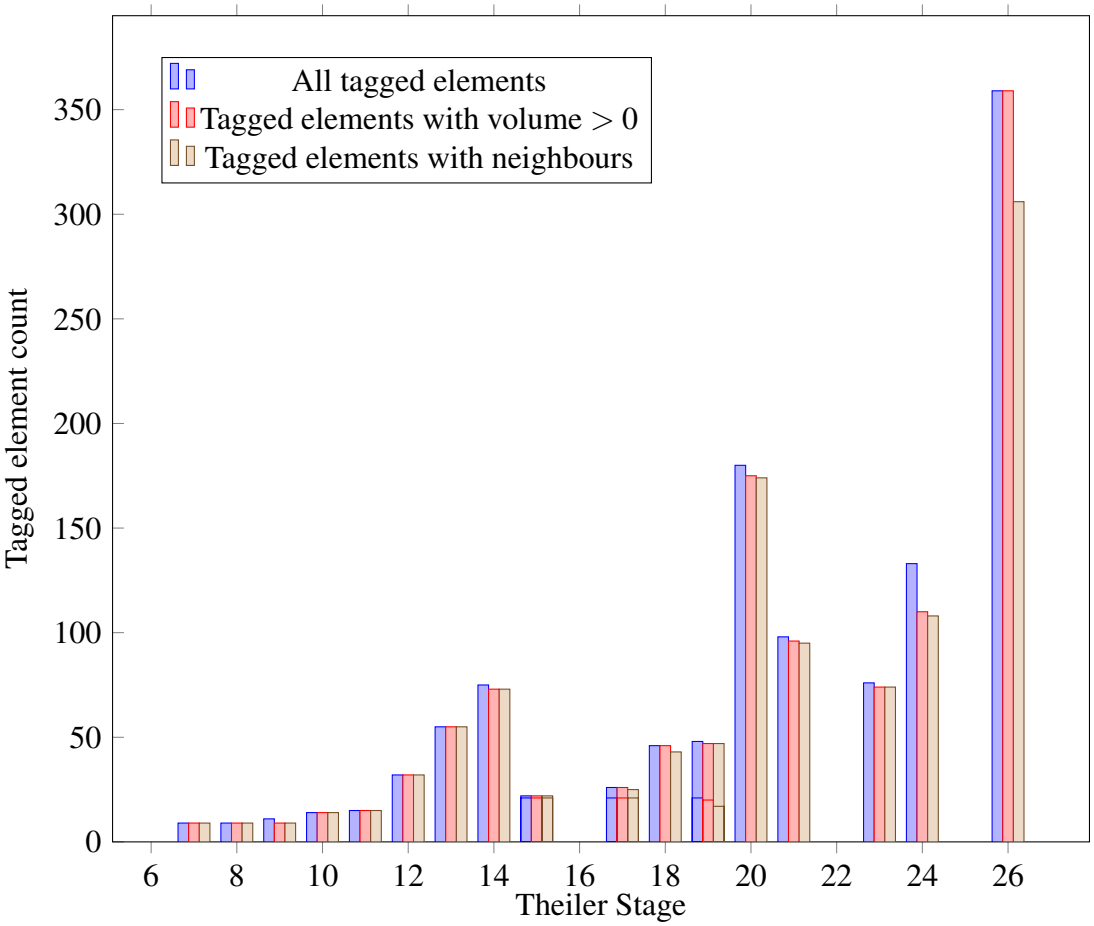


Figure 4.3: Basic statistics about datasets from the eMouseAtlas with anatomical delineations. Some datasets contain tagged elements with zero volume, or tagged elements which are not adjacent to any other. Elements with zero volume indicate a problem with the underlying data. For example EMA27 at Theiler Stage 14 has zero volume elements for the left and right umbilical veins.

It is evident that something unexpected is happening in Figure 4.3. It should not be the case that a mouse embryo loses anatomical diversity as it develops. The data for stages 15 through 19 and 21 through 25 seem particularly problematic. The explanation for this turns out to be quite mundane. The first stages were tagged manually, at significant cost, and resources were unfortunately not available to consistently continue this work (Baldock and Hill, personal communication). In some cases the latter stages

appear to have been tagged according to the particular interest of the researcher doing the work. This bias in the data is nevertheless interesting in understanding how to interpret our complexity measure in terms of intrinsic or extrinsic structure, which we discuss further below. Despite these defects in the data, we are able to obtain a signal, albeit a noisy one.

We have excluded several models from the following analysis. EMA149, at Theiler Stage 25, though it contains 78 delineated tissues, only four have non zero volume and only two have neighbours. Models EMA76, EMA103, EMA108 and EMA118 contain disconnected regions. This results in a q_{ij} that is not ergodic and therefore the Stationary Structural Entropy does not exist. Finally, EMA36 is an outlier suggesting a drastically different tissue delineation methodology. Its statistics are reported but excluded from the figures.

4.3.2 Structural Entropy of the eMouseAtlas

We now apply our Structural Entropy measure to the Mouse Atlas. Each stage has a different number of tagged elements. Since our goal is to quantify the degree of structure, for each stage, we compare the Structural Entropy (Equations 4.7 and 4.9) to the maximum possible value given by,

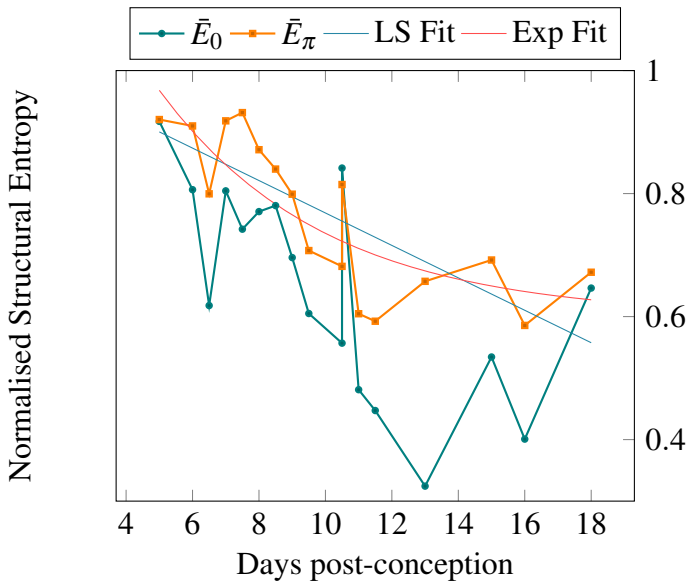
$$E_{\max} = -\log\left(\frac{1}{m}\right) \tag{4.14}$$

where m is the number of tagged elements. It is easy to see that as the number of tagged elements increases, the maximum entropy (degree of disorder) likewise increases. From this, we can define the normalised entropies,

$$\bar{E}_{\cdot} = \frac{E_{\cdot}}{E_{\max}} \tag{4.15}$$

which take on values from 0 to 1 and thus allow for comparison of the relative degree of disorder between developmental stages with different sets of tags. A value of 0 represents maximal structure, and 1 maximal disorder.

The results of this calculation are presented in Figure 4.4 and plotted against time measured in days post-conception. Two curves are shown, the one for $\bar{E}_0$, showing the amount of structure that is attributable purely to the volume distribution of tagged elements, with no account taken of their spatial relationships. The second curve, for $\bar{E}_\pi$, corresponds to the stationary distribution of the random walk among the tagged elements, as described above. The latter incorporates information about the volume through the q_{ii} as well as the spatial relationships through the q_{ij}, $i \neq j$.



Stage	Days
5	4
6	4.5
7	5
8	6
9	6.5
10	7
11	7.5
12	8
13	8.5
14	9
15	9.5
16	10
17	10.5
18	11
19	11.5
20	12
21	13
22	14
23	15
24	16
25	17
26	18

Figure 4.4: Normalised Structural Entropy as calculated for the eMouseAtlas data. Also shown is the least squares fit (LS Fit in the figure, with mean squared error 5.5×10^{-3}) for the Stationary Structural Entropy and a trial exponential fit (Exp Fit in the figure, mean squared error 4.5×10^{-3}). The table at right gives the correspondence between the Theiler stages present in the data and the time in days post-conception. Excluded from this figure is, $\bar{E}_0(\text{EMA36}) = 0.83$, $\bar{E}_\pi(\text{EMA36}) = 0.31$.

In both cases, we see a decreasing trend. This is interpreted as a decrease in disorder, or an increase in structure, as the mouse embryo develops. This signal is much clearer in the case of $\bar{E}_\pi$ which displays an orderly, almost linear decrease. Indeed a least squares fit for the normalised Stationary Structural Entropy has a mean squared error of 5.5×10^{-3} or two orders of magnitude smaller than the range of the entropy over the developmental phases covered by the dataset.

Clearly the decrease in disorder cannot be more than piece-wise linear as that would imply the nonsensical result that at some stage the organism becomes perfectly ordered with exactly one tissue as $E_\cdot \to 0$ and beyond to negative values of entropy which defy interpretation. A trial exponential fit is also shown, $\bar{E}_{\text{fit}} = e^{-0.2t} + 0.6$, that does not suffer from this problem of interpretation and has a mean squared error of 4.5×10^{-3}.

The data at early developmental stages bear closer inspection. While the general trend of our Stationary Structural Entropy measure, $\bar{E}_\pi$, is a steady decrease throughout the 13 days of development depicted in Figure 4.4, there is a short period, from days 7 to 8 (Theiler stages 10-11), in which $\bar{E}_\pi$ rises before returning to the trend. This period corresponds to one of the most remarkable events of metazoan development: gastrulation, when the primitive streak forms and cell movements in and through the epiblast transform the relatively orderly bilaminar disc into the three germ layers of the body. Gastrulation is widely regarded as being pivotal in development, Lewis Wolpert famously remarking that it is a life event more important than birth and marriage. It is interesting that this special stage of embryogenesis is detected by our tracking Stationary Structural Entropy over time.

To ascertain the extent to which the Structural Entropy calculation is biased by the number of tagged elements, we focus on a particular model, EMA27 from Theiler stage 14. This model contains 75 tagged elements of which 73 have non-zero volume. To understand how the Structural Entropy changes as the number of elements decreases, we merge adjacent elements. We do this by iterating through the list of elements, and merging between one and four neighbouring elements, chosen at random. We then calculate the Structural Entropy and Stationary Structural Entropy on this merged model.

We see that by randomly merging tagged elements, we introduce greater disorder. This is not unexpected. The original model was tagged in a particular way intended to correspond to an anatomical understanding of the embryo. This experiment takes no account of that, it simply merges elements that happen to be adjacent. With that done, both the Structural Entropy and the Stationary Structural Entropy are relatively

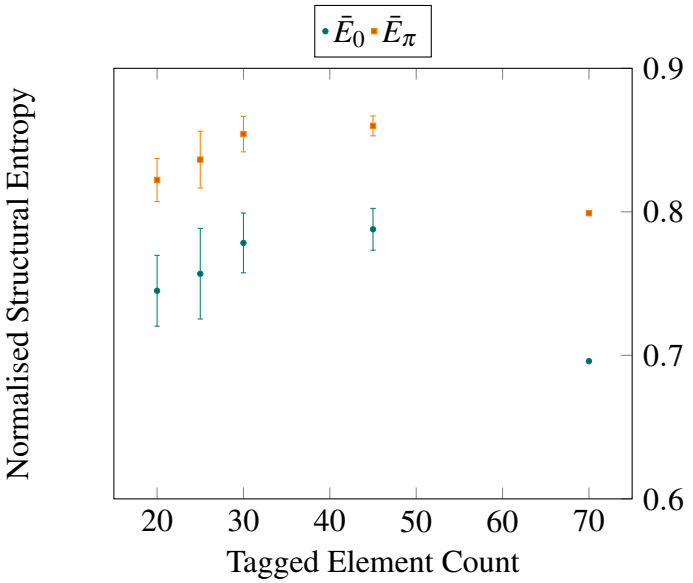


Figure 4.5: Normalised 0^{th} order and Stationary Structural Entropy for models created by merging tagged elements from EMA27, at Theiler stage 14. The data points isolated at the far right are for the original model. The merged models are created by merging at different depths: pairs, triples or quadruples of adjacent tissues. For each depth, 25 random models are generated and the resulting entropies are plotted according to the resulting number of tagged elements. The element count is discretised or grouped, e.g. 20-25 elements, 25-30 elements, and so forth. Error bars represent one standard deviation within a group.

stable with 30-60% of elements merged. Only when a clear majority of the elements are merged together do these measures change appreciably. In particular, we find a correlation of entropy and element count between 0.2 and 0.3, suggesting only a weak correlation between our measure and the absolute number of tagged elements.

4.4 Discussion

When Claude Shannon was discussing with John von Neumann what to call the quantity that came to be known as entropy in Information Theory, the latter famously quipped,

> You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage. (Tribus et al., 1971)

In an important sense, information theoretic entropy is an attributed quantity. It is a measure, as Shannon originally called it, of *uncertainty* about the state of a system. The trick that we have performed here is to define such a system: a particle moving at random through the organs of an embryonic mouse. We then suggested that our uncertainty about the whereabouts of the particle corresponds in some way to the structural complexity of the organism itself.

Tissues of different sizes contribute to our complexity measure in the following way. The measure is scale-independent in the sense that absolute tissue size plays no role. Embryos containing a given number of tissues, all of the same size, will have the same Structural Entropy regardless of their size. If the tissue sizes are different, the Structural Entropy will be correspondingly smaller. The degree of difference is the essence of order, to a first approximation. This is captured by the 0^{th} order measure, E_0, describing the role played by tissue volume alone.

To account for the spatial arrangement of tissues, we incorporate information about the connectivity between tissues. When we consider geometrically complex structures, an important feature is that their surface area is large compared to their volume. This large surface area means that the liminal region, or region of connectivity with adjacent tissues, is also larger. This is the reason that we claim that when we calculate the Stationary Structural Entropy, E_π, it captures this kind of structural complexity. More complex tissues “communicate” more with their neighbours and this, in turn contributes to a decrease in the Stationary Structural Entropy. The relative difference between

E_0 and E_π encodes the amount of organisation that can be attributed to the spatial arrangements as opposed to simply the amount of matter.

This approach may or may not be reasonable. We believe that it is, mainly because it accords with our intuition about what such order or structure ought to mean. It captures the sense that, despite the proliferation of tissues as the embryo develops, the organism becomes more ordered. If it did not, it would simply be a jumble of cells, an upper bound on disorder such as measured by Jamie A Davies (2016) using the taxonomy of cell types. That this is an upper bound is precisely what we see here: as development progresses, the Stationary Structural Entropy decreases relative to the equivalent disordered system, and it does so nearly consistently.

Another important aspect of the attributive nature of entropy arises from the data itself. In order to correctly compare like for like, each dataset should be tagged in the same way, using the same criteria. We have seen that there are defects in the data with some datasets processed meticulously and some processed more coarsely. Even if the data were consistently and meticulously processed it can be argued that measures such as Structural Entropy say more about the complexity of the underlying theoretical anatomical model than the intrinsic complexity of the body of the mouse. We can, however, only work with the data and theoretical tools that we have. By deriving randomly merged models we can see that our Structural Entropy measure is only weakly dependent on the absolute number of tagged elements.

The potential application of Structural Entropy to neuroscience, ageing and psychological disorders appears promising. Reus et al. (2014) considered the human brain connectome in an "edge-centric" as opposed to a "node-centric" way. In that article, communities of edges are identified; they seem to be significant but the meaning is left open: "The biological meaning of link communities in the brain is not immediately clear and very much open to scientific debate". The distinction between edge-centric and node-centric is reminiscent of that between E_0 and E_π above. De Reus' approach was applied as a measure of brain structure as a baseline in healthy elderly populations (Perry et al., 2015). Yeo et al. (2016) suggests that de Reus' approach may provide a useful indicator for psychological phenomena like schizophrenia, where differences were found, but it is unclear whether they are really significant or due to differences in methodology. There have also been some attempts to link it to general cognitive ability (Llufriu et al., 2017).

Voxel Based Morphometry (VBM) (J. Ashburner et al., 2000) is now a standard technique for comparing MRI scans tagged in a similar way to the anatomical data

that we have been considering. After some pre-processing: tagging, smoothing and registering images to the same spatial coordinates, the scans are compared voxel-wise. Among many applications this approach has been famously used to show plasticity in response to environmental demands (Maguire et al., 2000), that grey-matter normally decreases linearly with age (Good et al., 2001), and to ascertain the degree of progression of Alzheimer's disease (Testa et al., 2004; Matsuda, 2013). VBM shares some pre-processing requirements with what we can call Structural Entropy Morphometry (SEM), but then proceeds very differently. VBM is a calculation on voxels (or pixels in two dimensions) and SEM is explicitly not, it is concerned with the geometry of the tagged elements themselves. Crucially, VBM measures the relationship of scans from different groups whereas SEM is an intrinsic measure of the tagged object. Nevertheless it is plausible that SEM could recover the results of applying VBM and could yield additional insight. This possibility suggests potentially fruitful further research.

The concept, and ways of method for measuring, structural entropy can be applied to a wider range of problems than normal embryonic development. Much research attention is currently being expended on developing organoids – small structures made from stem cells that are intended to capture enough of the essence of a natural organ to be be useful for research (reviewed by Jamie A. Davies et al. (2018)). There is much debate within that field about how faithfully organoids, particularly organoids made by the different techniques of different laboratories, capture the complexity of the organ they are intended to represent. Structural entropy might be one useful measure. Another possible application is phylogeny: when discussing evolution, and particularly evolutionary developmental biology, it would be useful to have an objective measure of the anatomical complexity of adult organisms of different phyla or clades.

In this paper, we have called for the increased availability of high-quality tagged 3D datasets for the development of computational tools for anatomy. We have examined the eMouseAtlas dataset and produced some basic statistics about the tagging and annotation. We have extended Path Entropy to account for spatial structure and introduced Structural Entropy and studied the stationary distribution of a particle's random walk through tagged anatomical regions of developing mouse embryos. The stationary distribution illustrates clearly how the organism becomes more spatially structured as it develops. Finally applications of Structural Entropy Morphometry to neuroscience and the study of diseases related to ageing have been suggested as areas for future research.

4.5 Glossary

Edge A connection between two vertices on a graph (qv).

Entropy A measure of disorder: a highly ordered system (eg a perfectly alternating sequence of black and white tiles) has high entropy.

Graph A mathematical structure used to model pairwise relationships between objects. Graphs consist of "vertices" (the objects themselves) and "edges" (lines that connect them). In a model of a random walk, for example, the vertices might represent the spatial location of each footprint and the edges the strides that connect them.

Information Theory A field of science that focuses on the quantification, storage, retrieval and communication of information, particularly with relation to entropy.

Tag A tissue-type annotation associated with a spatial point on a digital model of an embryo. Eg point $(99, 65, 432)$ might have the tag *bladder urothelium*. Note that in Section 4.2.2, the word "colour" would usually be used in computer science or mathematics.

Vertex An elementary object in a graph (qv).

Chapter 5

Annotations for Rule-Based Models

Abstract. The chapter reviews the syntax to store machine-readable annotations and describes the mapping between rule-based modelling entities (e.g. agents and rules) and these annotations. In particular, we review an annotation framework and the associated guidelines for annotating rule-based models, encoded in the commonly used Kappa and BioNetGen languages, and present prototypes that can be used to extract and query the annotations. An ontology is used to annotate models and facilitate their description.

5.1 Why Annotations

The last decade has seen a rapid growth in the number of model repositories (Li et al., 2010; T. Yu et al., 2011; Snoep et al., 2003; Mısırlı; J. Hallinan, et al., 2014; Moraru et al., 2008). It is also well understood that the creation of models and of repositories requires expert knowledge and integration of different types of biological data from multiple sources (Endler et al., 2009). These data are used to derive the structure of, and

The work presented in Chapter 5 is adapted by permission from Springer Nature: *Modelling Biomolecular Site Dynamics* of *Methods in Molecular Biology* (ed. William Hlavacek), "Chapter 13: Annotations for Rule-Based Models" by Matteo Cavaliere, Vincent Danos, Ricardo Honorato-Zimmer William Waites (2019). The work was conceived by all of the authors and I drafted much of the chapter text. In particular, I contributed the discussion of reactions and rules, the relationship of annotations to the objects that are being annotated, and the relationship of the concepts of abstraction and annotation. This work was derived from a previous paper published in the journal *Bioinformatics* as "Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization" by Göksel Mısırlı, Matteo Cavaliere, William Waites, Matthew Pocock, Curtis Madsen, Owen Gilfellon, Ricardo Honorato-Zimmer, Paolo Zuliani, Vincent Danos and Anil Wipat (2015). I became involved with that paper at the beginning of my PhD studies after it had been returned from *Bioinformatics* for major revisions. My contributions were the method of adding annotations in concrete rule-based languages in a backwards compatible way, implementation of the `krdf` software for extracting and processing these annotations, writing queries to demonstrate how useful information can be extracted from the annotations, generation of the contact map diagram and significant portions of the final text.

parameters for, models. However which data is used and how the model is derived from that data is not part of the model unless we explicitly annotate it in a well-defined way.

In general, annotations decorate a model with metadata linking to biologically relevant information (Michael L Blinov; Ruebenacker, et al., 2010). They can facilitate the automated exchange, reuse and composition of complex models from simpler ones.

Annotations can be used to aid in the computational conversion of models into a variety of other data formats. For example, PDF documents (Li et al., 2010) or visual graphs (Funahashi et al., 2007) can be automatically generated from annotated models in order to aid human understanding.

On the computational and modelling side, rule-based languages such as Kappa (Danos; Laneve, 2004; Danos; Jérôme Feret; W. Fontana; Krivine, 2007) and BioNetGen (James R. Faeder et al., 2009) have emerged as helpful tools for modelling biological systems (Köhler et al., 2014). One of the key issues is that rule-based modelling is used to concisely represent the combinatorial explosion of the state space inherent in modelling biological systems.

These types of modelling languages have facilities to add comments that are intended for unstructured documentation and usually directed at the modeller or programmer. These comments are in general human and not machine-readable. This can be a problem because the biological semantics of the model entities are not computationally accessible and cannot be used to influence the processing of the models.

There are other previous works that have addressed the issue of annotations in rule-based models. In particular, (Chylek; Hu, et al., 2011) suggest extending rule-based models to include metadata, focusing on documenting models with biological information using comments to aid the understanding of models for humans. More recently, in (Klement et al., 2014) authors have presented a way to add data in the form of property/value pairs using a specific syntax. On the other hand, machine-readable annotations have been applied to rule-based models using PySB, a programming framework for writing rules using Python (Lopez et al., 2013a). However, this approach is restricted as annotations cannot be applied to sites or states.

In this chapter we first discuss the general idea of annotation, its relation with the concept of abstraction and then review the proposal of the annotation framework for rule-based models as recently introduced and defined by Mısırlı; Cavaliere, et al. (2015).

5.2 Reactions, Rules, Annotations and Abstractions

Before entering into the technicalities of the presented annotation framework, we would like to discuss in an informal and intuitive manner the differences between models created using reactions versus those obtained using rules, discussing the advantages of considering annotations and how they are strictly linked to the much more general notion of abstraction.

5.2.1 Reactions and Rules

Rules as they are to be understood in the present context are a sort of generalisation of reactions of the type familiar from chemistry. The reason this generalisation is useful can be easily seen. Consider the following toy examples,

$$\circ + \circ \rightarrow \circ\text{–}\circ$$

which can be understood as a step in the creation of a polymer from two monomers. Multiple applications of this rule result in a progressively longer chain of molecules,

$$\circ\text{–}\circ + \circ \rightarrow \circ\text{–}\circ\text{–}\circ$$

$$\circ\text{–}\circ\text{–}\circ + \circ \rightarrow \circ\text{–}\circ\text{–}\circ\text{–}\circ$$

$$\dots$$

Writing this down with traditional reaction notation, we would be forced to explicitly generate the entire unbounded sequence of reactions with an unbounded number of chemical species,

$$A + A \rightarrow A_2$$

$$A_2 + A \rightarrow A_3$$

$$A_2 + A_2 \rightarrow A_4$$

$$A_3 + A \rightarrow A_4$$

$$\dots$$

Clearly this is unworkable for computation. The solution is to allow a species to have *sites* at which connections can be made. In the above example, the species could be described as `A(u,d)`, that is substance `A` with an upstream and a downstream site. The interaction can then be written as,

$$\texttt{A(d)},\ \texttt{A(u)} \rightarrow \texttt{A(d!1)},\ \texttt{A(u!1)}$$

where the notation `d` means that the downstream site is unbound, and the `!1` means bound with a particular edge. Note that this says nothing about the state of the upstream site in the first instance of `A` nor the downstream in the second, so there can be an arbitrarily long chain of molecules attached at those sites. It is easy to see that this compact notation captures both the infinite sequence of reactions and the infinite set of species that would be required to express the same interaction as a chemical reaction.

5.2.2 Annotations

Informally, the word "annotation" has a meaning similar to "documentation" but with a difference in scale. Whereas documentation connotes a rather large text describing some thing, annotation is expected to be much shorter. It also evokes proximity: it should be in some sense "near" or "on" the thing being annotated. In both cases there seems to be a sharp distinction between the text and its object. The object should exist in its own right, be operational or functional in the appropriate sense without need to refer to extraneous texts. Annotation might help to understand the object but the object can exist and work just fine on its own.

This folk theory of annotation breaks down almost immediately under inspection. A typical example is data about a book such as might be found in a library catalogue. This is actually the canonical example used to explain what is meant by *metadata* or data about data. The first observation is that if we look at a book and peruse the first few pages it is almost certain that we will find information about who wrote it and where and when it was published. This information is not the book, it is metadata about the book, but it is contained within the covers of the book itself.

Perhaps this is not so serious a problem. It is possible in principle to imagine that a book, say with the cover and first few pages torn out, is still a book that can be read and enjoyed. Perhaps somehow the metadata is *separable* and that is the important idea. The book part of the artefact can exist on its own and serve its purpose independently of any annotation or metadata that while it might usually be found attached, can easily be removed without affecting the fundamental nature of the thing. But what of other things that we might want to do with a book?

A favourite activity of academics is *citing* documents such as books and journal articles. This means including enough information in one work to unambiguously *refer* to another. There is an urban legend that Robarts Library at the University of Toronto is said to be sinking because the engineers charged with building it did not account for

the weight of the books within. Supposing that this were true, these poor apocryphal engineers could have used metadata within the university's catalogue to sum up the number of pages of all the books and estimate their weight to prevent this tragedy.

More mundanely, categorising and counting books in order to plan for the use of shelf space in a growing collection, or even locating a book in a vast library seem to be a plausible things to do with metadata that do not involve any actual books. Manipulation and productive use of annotation is possible in the absence of the objects and well-defined even if the objects no longer exist. One imagines the despondent librarians and archivists of Alexandria making such lists to document and take stock of their losses after the great fire.

Now suppose that this list created by the librarians of Alexandria itself ended up in a collection in some other library or museum. It is given a catalogue number, the year it was acquired is marked. Now what was metadata has now itself become the object of annotation! And here we arrive at the important insight: what is to be considered annotation and what is to be considered object depends on the purpose one has in mind for it. If the interest is the collection of books in Alexandria, the list is metadata, a collection of annotations, about them. If the interest is in the documents held by a contemporary museum, among which the list is to be found, the list is an object. The distinction is not intrinsic.

Turning to the subject at hand, the objects to be annotated are rules. According to the folk theory of annotation, there should be a sharp distinction between rules and their annotation. When it comes to executing a simulation, the software that does this need not be aware of the annotations. Indeed the syntax for annotating rules described here is specifically designed for backwards compatibility such that the presence of annotations should not require any disruption or changes to existing simulation software.

So long as the *purpose* of the annotations is as an aid to understanding the rules the location of the distinction between rule and annotation is fixed in this way. The obvious question is, are there other uses to which the annotations can be put?

In the paper where the presented annotation mechanism was first described ((Mısırlı; Cavaliere, et al., 2015)) one of the motivating examples was to create a *contact map*, a type of diagram that shows which agents or species interact with each other and labels these interactions with the rule(s) implementing them (an example of contact-map is also recalled later in this chapter, see Figure 5.6).

The contact map use is illustrative of how movable the separation between object and annotation is (Buneman et al., 2013). The entities of interest, rules and agents, are

on the one hand decorated with what seems to be purely metadata: labels, or friendly human-readable names that are suitable for placing on a diagram, preferable to the arbitrary machine-readable tokens that are used by the simulator (arbitrary because they are subject to renaming as required). On the other hand, the interactions between the substances, what we wish to make a diagram *of*, are written down in a completely different language with an incompatible syntax.

A minor change of perspective neatly solves this problem. It is simply to rephrase the rule, saying "A and B are related, and the way they are related is that they combine to form C". This has the character of annotation: the rule itself is a statement about the substances involved. More particularly it describes a *relation* between the substances. On close inspection, giving a token used in a rule a human-readable name is also articulating a relation, that is the relation called "naming" between the substance and a string of characters suitable for human consumption.

With this change of perspective, all of the information required to make the diagram is now on the same level. The only construct that must be manipulated is sets of relations between entities (and strings of text, which are themselves a kind of entity). Fortunately there exist tools and query languages for operating on data stored in just this form. Having worked out the correct query to extract precisely what is needed to produce the diagram, actually doing so is trivial.

5.2.3 Abstractions and Annotations

In the preceding section on annotation, much was made of a "movable" line. "Above" this line are annotations and "below" it are the objects. The sketch of a procedure for producing a diagram to help humans understand something about a system of rules as a whole illustrated that it can be convenient to place this line somewhere other than might be obvious at first glance—and this example will be worked in more detail below to demonstrate how this happens in practice. However the idea of such a line and how it might be moved and what exactly that means is still rather vague. Let us now make this notion more precise.

Formally, a *relation* between two sets, X and Y is a subset of their Cartesian product, $X \times Y$. In other words it is a set of pairs, $\{(x,y) \,|\, x \in X, y \in Y\}$, and it is usually the case that it is a proper subset in that not all possible pairs are present in the relation. In order to compute with relations, the sets must be *symbols*, $X, Y \subseteq \mathbb{S}$, ultimately sequences of bits because a computer or Turing machine is defined to operate on such sequences

and not on every day objects such as books, pieces of fruit, molecules or sub-atomic particles, or indeed concepts and ideas.

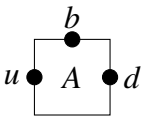
This last is an important point. It is not possible to compute with objects in the world, be they concrete or abstract, it is only possible to compute with *representations of* these objects. Another kind of relation is required for this, $\mathbb{R} \subseteq \mathbb{S} \times \mathbb{W}$ where $\mathbb{W}$ is the set of objects in the world. It is not possible to write down such relations between symbols and real-world objects any more than it is possible to write down an apple. So we have two kinds of relations to work with: annotations which are relations among symbols and live in $\mathbb{S} \times \mathbb{S}$ and representations which map between symbols and the world, $\mathbb{S} \times \mathbb{W}$.

Some observations are in order. First, the representation relation has an inverse, $\mathbb{W} \times \mathbb{S}$. This is trivial and is simply "has the representation" as opposed to "represents". Second, of course, symbols are themselves objects in the world, so $\mathbb{S} \subset \mathbb{W}$. Finally, relations among symbols—annotations — are likewise objects in the world, so $\mathbb{S} \times \mathbb{S} \subset \mathbb{W}$ also. This is useful because it means that it is possible to represent annotations with symbols and from there articulate relationships among them using more annotations, constructing a hierarchy of annotation as formalised by Buneman et al. (2013). We run into trouble though if we try to say that representations are in the world because $\mathbb{S} \times \mathbb{W} \not\subset \mathbb{W}$, and this is why they cannot be written down. Symbols represent, annotations are relations among symbols, and the character of representation is fundamentally different from that of annotation.

Now we have enough to explain the intuition behind the folk theory of annotation, that there is a difference of kind between the annotation and its object. This difference is just the same as considering a notional pair $(x \in \mathbb{S}, -)$ *qua* annotation or *qua* representation, that is, deciding the set from which the second element of the tuple should be drawn. A similar choice is available, *mutatis mutandis*, for the inverse, $(-, x \in \mathbb{S})$. If unspecified element is in $\mathbb{W} \backslash \mathbb{S}$, those objects in the world that are not symbols, there is only one choice: the relation can only be treated as representation. If it is in $\mathbb{W} \cap \mathbb{S}$ then either interpretation is possible, and one or the other might be more appropriate depending on the purpose or question at hand.

The ability to make this choice is the ability to select an appropriate *abstraction*. Selecting an abstraction means deciding to interpret a relation as representation and not annotation. This is best illustrated with an example. Here is a (representation of an)

agent or substance,



$\mathtt{A(u,d,b)}$

Perhaps it is a fragment of DNA which can be connected up-stream and down-stream to other such fragments, and it has a binding site where RNA-polymerase can attach as part of the transcription process. Some annotations involving A might be,

$$(A, \text{“Promoter”}) \in \mathbb{L}$$
$$(A, \mathtt{TTGATCCCTCTT}) \in \mathbb{M}$$

where the first is from the set of labellings, $\mathbb{L}$ and the second is from the set of correspondences with symbols representing nucleotide sequences which we will call $\mathbb{M}$. A more conventional way of writing these corresponding more closely to the Semantic Web practice is,

```
A label "Promoter" .
A has sequence TTGATCCCTCTT .
```

The labelling annotation is easy to understand. It simply provides a friendly string for humans.

The second annotation is more challenging. It says that the DNA fragment represented by A corresponds to a certain sequence of nucleotides. On the one hand the symbol for that sequence could simply be taken as-is, since it does not play an explicit role in the computer simulation of whatever interactions A is involved in. That corresponds to treating the symbol `TTGATCCCTCTT` as a representation. It is the end of the chain, there only remains the relation from that symbol to something in the world, which is not something that we would wish to write down or compute with.

On the other hand, it is equally possible to write down an annotation on the sequence symbol that specifies the list of (symbols representing) the nucleotides that it consists in,

```
TTGATCCCTCTT consists [T,T,G,A,T,C,C,C,T,C,T,T] .
```

Such a verbose formulation might be useful if one had, for example, a machine for synthesizing DNA molecules directly to implement an experiment *in vitro* for a genetic circuit that had already been developed and tested by simulation *in silico*. In this case the symbols, `A`, `C`, `T` and `G` play the role of representing real-world objects and the symbol `TTGATCCCTCTT` is merely a reference that can be used to find the (list-structured)

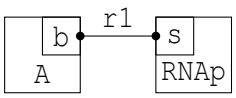
relations among them. By making this choice, the selected abstraction has become more granular.

Another example, pertinent because while we do not yet have machines for arbitrarily assembling DNA molecules from individuals, we do have tools for drawing contact map diagrams which we will use later, is a rule involving this agent. This agent has a binding site which may be occupied by an RNA-polymerase molecule at a certain rate. This could be expressed as,

```
#^ r1 label "Binding of RNAp to A"
'r1' A(b!_), RNAP(s!_) -> A(b!1), RNAP(s!1) @k
```

where now we have introduced a little bit more of the syntax that will be more fully elaborated later for annotating rules written in a file using the Kappa language. Here a rule is simply given a useful human-readable label, the canonical example of annotating something. On its own, it is useful, imagine a summary of the contents of a set of such rules using labels like this. For that purpose the symbol `r1` can be considered just to represent the rule without looking any deeper.

For a contact map diagram, more information is needed. At right is the diagram that corresponds to the example rule. It shows that `A` and `RNAp` interact, that it happens through the action of the rule `r1` and in particular involves the sites `b` and `s`. Perhaps including which sites are involved in the interaction is too granular and it might be desireable in some circumstances to have a similar diagram involving just the agents and the rules. Or perhaps more information is desired to be presented in the diagram such as whether the rule involves creation or annihilation of a bond, say using arrows or a broken edge. No matter the level of granularity required, it is clear that the necessary information is contained within the rule itself, so simply considering the symbol `r1` to represent to the rule *qua* rule is not enough. Such a level of abstraction would be too coarse, it must be elaborated further. Instead it should be considered to represent annotations that themselves represent the structure of the rule.



To elaborate the rule sufficiently to support the production of such a diagram involves a much greater amount of annotation structure than we have seen so far. A rule has a left and a right side. Each of those has zero or more agent *patterns*. A rule does not involve agents as such, rather it involves patterns that can match configurations of agents, so patterns then relate, *intra alia*, to agents and sites, and finally bonds between sites that are either to be matched (on the left-hand side) or created or annihilated (on the right-hand side).

5.3 Rule-Based Models: Brief Introduction

After the general discussion on the role of annotations and rule-based models, in this Section we move to the more technical aspects (focusing on two languages, Kappa and BioNetGen) and follow the terminology and the definitions provided in (Mısırlı; Cavaliere, et al., 2015).

Biological entities are represented by agents in Kappa and molecule types in BioNetGen (we use 'agent' to generically refer to both types). Agents may include any number of sites that represent the points of interactions between agents. For example, the binding domain site of a transcription factor (TF) agent can be connected to a TF binding site of a DNA agent. Moreover, sites can have states. For instance, a TF could also have a site for phosphorylation and the DNA binding can be constrained to occur only when the state of this site is phosphorylated.

For an agent with two sites, of which one with two internal states and the other with three, the number of possible combinations is six (Figure 5.1A, B). A pattern is an (possibly incomplete) expression of an agent in terms of its internal, and binding states. Rules, that specify biological interactions, consist of patterns on the left hand side which, when match, produce the result on the right hand side (Figure 5.1C). Specific patterns of interest can be declared as an observable of the model.

It is important to highlight that while the syntactic definition of an agent identifies sites and states in rule-based models, the semantics of sites and states is usually clear only to the modeller. Cleary, if one wishes to have machine access, then this information must be exposed in a structured way. The key idea of the approach presented in (Mısırlı; Cavaliere, et al., 2015) and that we review in what follows, is to extend the syntax of rule-based models to incorporate annotations.

Existing metadata resources include machine readable controlled vocabularies and ontologies, Web services providing standard access to external identifiers and guidelines for the use of these resources. For example, the Minimum Information Requested in the Annotation of Models (MIRIAM) standard (Le Novère et al., 2005) propose the standard minimal information required for the annotation of models.

Following (Mısırlı; Cavaliere, et al., 2015) we suggest that entities in models are linked to external information through the use of unique and never ambiguous Uniform Resource Identifiers (URIs), which are embedded within models. The uniqueness and global scope of these URIs are then crucial for *disambiguation* of model agents, variables and rules.

```
A: An agent definition
A(site1~u~v, site2~x~y~z)

B: Possible combinations of internal states
A(site1~u,site2~x)
A(site1~u,site2~y)
A(site1~u,site2~z)
A(site1~v,site2~x)
A(site1~v,site2~y)
A(site1~v,site2~z)

C: An example binding rule
A(site1~v,site2~z),A(site1~v,site2~y)
    -> A(site1~v!1,site2~z),A(site1~v!1,site2~y) @kf
```

Figure 5.1: **A.** An agent with two sites. `site1` has two possible internal states while `site2` has three. **B.** This agent can be used in six different ways depending on the internal states of its sites. **C.** A rule that specifies how agent `A` forms a dimer when the state of `site1` is `v` and the states of `site2` are `z` and `y`, respectively. The symbol `!n` means that the sites where it appears are bound (connected) together. The constant `kf` denotes the kinetic rate associated with the rule.

We also choose to represent annotations using the Resource Description Framework (RDF) data model (Cyganiak et al., 2014; Gandon et al., 2014) as statements or binary predicates. A statement can link a modelling entity to a value using a standard qualifier term (predicate), which represents the relationship between the entity and the value. These qualifiers often come from controlled vocabularies or ontologies in order to unambiguously identify the meaning of modelling entities. URIs are used as values to link these entities to external resources, and hence to a large amount of biological information by keeping the amount of annotations minimal. The links themselves are typed, again with URIs. The qualifiers and resources to which they refer are drawn from ontologies that encode the Description Logic (Harmelen et al., 2004) for a particular domain.

Semantics can be unified by means of metadata with controlled vocabularies. There are several metadata standard initiatives that provide controlled vocabularies from which standard terms can be taken. For instance, metadata terms provided by the Dublin Core Metadata Initiative (DCMI) (DCMI Usage Board, 2012) or BioModels qualifiers can be used to describe modelling and biological concepts (Le Novère et al., 2005; Li et al., 2010). On the other hand, ontologies such as the Relation Ontology provide formal

definitions of relationships that can be used to describe modelling entities (B. Smith et al., 2005). There are also several other ontologies and resources that are widely used to classify biological entities represented in models with standard values (Swainston et al., 2009): the Systems Biology Ontology (SBO) (Courtot et al., 2011) to describe types of rate parameters; the Gene Ontology (GO) (The Gene Ontology Consortium, 2001) and the Enzyme Commission numbers (Bairoch, 2000) to describe biochemical reactions; the Sequence Ontology (SO) (Eilbeck et al., 2005) to annotate genomic features and unify the semantics of sequence annotation; the BioPAX ontology (Demir et al., 2010) to specify types of biological molecules and the Chemical Entities of Biological Interest (ChEBI) (Degtyarenko et al., 2008) terms to classify chemicals. URIs of entries from biological databases, such as UniProt (Magrane et al., 2011) for proteins and KEGG (Kanehisa et al., 2008) for reactions, can also be used to uniquely identify modelling entities.

The access to the data should be unified in the sense that well-known identifiers should be used so that an entity can be referred to independently in different models, or works or bodies of research. This can be done by accessing external resources through URIs using MIRIAM or Identifiers.org URIs (Juty et al., 2012), The former is not directly dereferencable, which means that it does not contain, built-in, enough information to retrieve more information about the entity that it denotes. It requires out of band knowledge to retrieve information, the knowledge of where additional information about the MIRIAM identitifiers is kept and how it can be retrieved. By contrast, Identifiers.org provides a standard HTTP mechanism for finding more information and should be preferred for this reason. These URIs refer to collections of entities and terms representing the entities themselves. For example, the MIRIAM URI `urn:miriam:uniprot:P69905`[3] and the Identifiers.org URI `http://identifiers.org/uniprot/P69905` can be used to link entities to the P69905 entry from UniProt. The relationships between modelling entities, annotation qualifiers and values can be represented using RDF graphs.

We suggest to use RDF syntax that represents knowledge in the form of (subject, predicate, value) triples, in which the subject can be an anonymous reference or a URI, the predicate is a URI and the object can be a literal value, an anonymous reference or a URI.

Subjects and objects may refer to an ontology term, an external resource or an

[3] A dereferenceable URI using the MIRIAM Web services is `http://www.ebi.ac.uk/miriamws/main/rest/resolve/urn:miriam:uniprot:P69905`

entity within the model. RDF graphs can be then serialized in different formats such as XML or the more human readable Turtle format(Prud'hommeaux; Carothers, 2014). Modelling languages such as the Systems Biology Markup Language (SBML) (Hucka et al., 2003), CellML (Cuellar et al., 2003; Hedley et al., 2001) and Virtual Cell Markup Language (Moraru et al., 2008) are all XML-based and provide facilities to embed RDF/XML annotations (Endler et al., 2009).

Moreover, there are also other exchange languages, such as BioPAX and the Synthetic Biology Open Language (SBOL) (Galdzicki; Wilson, et al., 2012; Galdzicki; Clancy; Oberortner; Pocock; J. Quinn, et al., 2014), that can be serialised in RDF/XML allowing custom annotations to be embedded.

Following the suggestion of (Mısırlı; Cavaliere, et al., 2015) one can extend the use of RDF and MIRIAM annotations to describe a syntax to store machine-readable annotations and an ontology to facilitate the mapping between rule-based model entities and their annotations. We illustrate annotations using terms from this ontology and propose some examples.

5.4 Annotations for Rule-Based Models

In this Section we review the syntax originally defined in (Mısırlı; Cavaliere, et al., 2015) for storing annotations.

We start by noticing that a common approach, when trying to add additional structured information to a language where it is undesirable to change the language itself, is to define a special way of using comments. This practice is established for structured documentation or "docstrings" in programming languages (Acuff, 1988; Stallman et al., 1992).

The idea is to use this same approach so that models written using the conventions that we describe here do not require modification of the modelling software, KaSim (Krivine et al., 2018) and RuleBender (W. Xu et al., 2011), that is their primary target.

For this reason, we use the language's comment delimiter followed by the '^' character to denote annotations in the textual representation of rule-based languages. Kappa and BioNetGen use the '#' symbol to identify comment lines, so in the case of these languages, comments containing annotations are signalled by a line beginning with '#^'. This distinguishes between comments containing annotations and comments intended for human consumption. Annotation data for a single modelling entity or a

model itself can be declared over several lines and each line is prefixed with the '`#^`' symbol.

Annotations are then serialised in the RDF/Turtle format, that leads to a good balance between the need for a machine-readable syntax and a human readable textual representation. Rule-based modelling languages are themselves structured text formats designed for this same balance, so RDF/Turtle is more suitable than the XML-based representations of RDF.

Annotations for a single rule-based model entity are a list of statements. It is important to stress that annotations may refer to other annotations within the same model. When all the lines corresponding to a rule-based model and the annotation delimiter symbols are removed, the remaining RDF lines can represent a single RDF document. This enables annotations to be quickly and easily extracted without special tools[4].

In textual rule-based models, it is difficult to store annotations within a modelling entity since Kappa and BioNetGen represent modelling entities such as agents and rules as single lines of text. As a result, there is no straightforward location to attach annotations to an entity. Following (Mısırlı; Cavaliere, et al., 2015) we achieve the mapping between a modelling entity and its annotations by defining an algorithm to construct a URI from the symbol used in the modelling language. The algorithm generates unique and unambiguous prefixed names that are intended to be interpreted as part of a Turtle document. The algorithm simply constructs the local part of a prefixed name by joining symbolic names in the modelling language with the ':' character, and prepending the empty prefix, ':'. This means that one must satisfy the condition that the empty prefix is defined for this use. Using this algorithm, we can derive a globally unique reference for the `y` internal state of site `site2` of agent `A` from `A(site1~u~v,site2~x~y~z)` as `:A:site2:y`.

In Kappa, rules do not have symbolic names but each rule can be preceded by free text surrounded by single quotes. Such free text should be consistent with the local name syntax in Turtle and SPARQL (Prud'hommeaux; Seaborne, 2013) languages. If that is satisfied, identifiers for subrules are created by just adding their position index, based on one, to the identifier for a rule (see Figure 5.4B). A similar restriction is placed on other tokens used in the models; agent and site names, variable and observable names

[4]For example, on a UNIX system, the following pipeline could be used:

```
grep '^#\^'| sed 's/^#\^//'
```

must all conform to the local name syntax.

Controlled vocabularies such as BioModels.net qualifiers are formed of *model* and *biology* qualifiers. The former offers terms to describe models. BioModels.net qualifiers are also appropriate to annotate rule-base models, but additional qualifiers are needed to fully describe rule-based models. These are specific to the annotation of rule-based models and this is done by using a distinct ontology—the *Rule-Based Model Ontology*—in the namespace `http://purl.org/rbm/rbmo#` conventionally abbreviated as `rbmo` (we omit the prefix if there is no risk of ambiguity). Each qualifier is constructed by combining this namespace with an annotation term. A subset of significant terms are listed in Appendix A.2 (presented in the appendix) while the full ontology is available online at the namespace URI.

The `Model` classes such as `Kappa` and `BioNetGen` specify the type of the model being annotated. The term `Agent` is used to declare physical molecules. Hence, the `Agent` class can represent agents and tokens in Kappa, or molecule types in BioNetGen. `Site` and `State` represent sites and states in these declarations respectively. Rules are identified using `Rule`. The predicates `hasSite` and `hasState` and their inverses are used to annotate the links between agents, sites and internal states declarations. Appendix A.2 reviews the terms related to the declaration of the basic entities from which models are constructed. We assume that the terms that start with an uppercase letter are types (In the sense of `rdf:type`, and also in this instance `owl:Class`) for the entities in the model which the modeller could be expected to explicitly annotate. The predicates begin with a lowercase letter and are used to link entities to their annotations.

Appendix A.3 includes terms to facilitate representation of rules in RDF. This change of representation (materialization), from Kappa or BioNetGen to RDF is something that can easily be automated and a tool is already available (for models written in Kappa).

This representation in RDF is helpful for analysis of models because merges the model itself with the metadata in a uniform way easy to query. Annotations that cannot be derived from the model (as well as the model itself) are written explicitly in RDF/Turtle using the terms from Appendix A.2 embedded in comments using a special delimiter. Extra statements can then be derived by parsing and analyzing the model using terms from Appendix A.3 and the same naming convention from the algorithm previously described. These statements are then merged with the externally supplied annotations to obtain a complete and uniform representation of all the information about the model.

The open-ended nature of the RDF data model means that it is possible to freely

incorporate terms from other ontologies and vocabularies, including application-specific ones. In this respect, two terms are crucial. The `dct:isPartOf` predicate from DCMI Metadata Terms is used to denote that a rule or agent declaration *is part of* a particular model (or similarly with its inverse, `dct:hasPart`).

The `bqiol:is` predicate from the *Biomodels.net Biology Qualifiers* is used to link internal states of sites to indicate their biological meaning. This term is chosen because it denotes a kind of identification that is much weaker than the logical replacement semantics of `owl:sameAs`. Using the latter would imply that everything that can be said about the site *qua* biological entity can also be said about the site *qua* modelling entity. Clearly, these are not the same and identifying them in a strong sense would risk incorrect results when computing with the annotations.

Appendix A.1 enumerate useful ontologies and vocabularies with their conventional prefixes to annotate rule-based models. This list is not exhaustive and can be extended.

5.5 Adding Annotations to Rule-Based Models

In this Section we demonstrate how the suggested annotations can be added to rule-based models. Again we follow the methodology originally presented in (Mısırlı; Cavaliere, et al., 2015).

```
#^@prefix : <http://.../tcs.kappa#>.
#^@prefix rbmo: <http://purl.org/rbm/rbmo#>.
# ... other prefixes elided ...
#^@prefix dct: <http://purl.org/dc/terms/>.
#^@prefix foaf: <http://xmlns.com/foaf/0.1/>.

#^ :kappa a rbmo:Kappa  ;
#^   dct:title "TCS_PA Kappa model" ;
#^   dct:description
#^     "Two component systems and promoter architectures" ;
#^   dct:creator "Goksel Misirli", "Matteo Cavaliere";
#^   foaf:isPrimaryTopicOf <https://.../tcs.kappa> .
```

Figure 5.2: An example model annotation (as in Mısırlı; Cavaliere, et al. (2015)), with details about its name, description, creators and online repository location. The prefix definitions required to annotate the model are defined first, and the empty prefix is defined for the model namespace itself.

```
A:
#^:ATP a rbmo:Agent ;
#^  bqbiol:isVersionOf chebi:CHEBI:15422 ;
#^  biopax:physicalEntity biopax:SmallMolecule .
%token: ATP()
B:
#^:Kinase a rbmo:Agent ;
#^  rbmo:hasSite :Kinase:psite ;
#^  bqbiol:is uniprot:P16497 ;
#^  biopax:physicalEntity biopax:Protein ;
#^  ro:hasFunction go:GO:0000155 .
#^:Kinase:psite a rbmo:Site ;
#^  rbmo:hasState :Kinase:psite:u, :Kinase:psite:p .
#^:Kinase:psite:u a rbmo:State ;
#^  bqiol:is pr:PR:000026291 .
#^:Kinase:psite:p a rbmo:State ;
#^  bqiol:is psimod:MOD:00696 .
%agent: Kinase(psite~p~u)
C:
#^:pSpo0A a rbmo:Agent ;
#^  rbmo:hasSite :pSpo0A:tfbs ;
#^  bqbiol:isVersionOf so:SO:0000167 ;
#^  biopax:physicalEntity biopax:DnaRegion ;
#^  sbol:nucleotides "ATTTTTTTAGAGGGTATATAGCGGTTTTGTCGAATGTAAACATGTAG" ;
#^  sbol:annotation :pSpo0A_annotation_28_34 .
#^:pSpo0A:tfbs a rbmo:Site ;
#^  bqbiol:isVersionOf so:SO:0000057 ;
#^  biopax:physicalEntity biopax:DnaRegion ;
#^  sbol:nucleotides "TGTCGAA" .
#^:pSpo0A_annotation_28_34 a sbol:SequenceAnnotation ;
#^  sbol:bioStart 28;
#^  sbol:bioEnd 34 ;
#^  sbol:subComponent :pSpo0A:tfbs .
%agent: pSpo0A(tfbs)
D:
#^:Spo0A a rbmo:Agent .
%agent: Spo0A(psite~p~u)
#^:Spo0A_p a rbmo:Observable ;
#^   ro:has_function go:GO:0045893 .
%obs: 'Spo0A_p' Spo0A(psite~p)
```

Figure 5.3: Examples of agent annotations for **A.** An ATP token agent. **B.** A kinase agent with phosphorylated and unphosphorylated site. **C.** A promoter agent with a TF binding site. **D.** An agent and an associated observable for the phosphorylated Spo0A protein, which can act as a TF.

Annotations are added by simply adding a list of prefix definitions representing annotation resources providing relevant terms for the annotation of all model entities (such as agents and rules). These definitions are followed by statements about the title and description of the model, using the `title` and `description` terms from *Dublin Core*. Annotations can be expanded to include model type, creator, creation time, its link to an entry in a model database (Figure 5.2).

Appendix A.4 lists predicates, both from the `rbmo` ontology and from other vocabularies, for annotating distinct entities in a model. Figure 5.3 shows examples of Agent annotations. In Figure 5.3A the ATP token is annotated as a small molecule with the id of 15422 from CHEBI. Agents without sites can also be annotated in a similar way. In Figure 5.3B, the agent is specified to be a protein using the `biopax:Protein` value for the `biopax:physicalEntity` term. This protein agent is annotated as P16497 from UniProt, which is a kinase (i.e. an enzyme that phosphorylates proteins) involved in the process of sporulation. It has a site with the phosphorylated and unmodified states, which are annotated with corresponding terms from the Protein Modification Ontology (Montecchi-Palazzi et al., 2008).

The `ro:hasFunction` term associates the agent with the GO's histidine kinase molecular function term `GO:0000155`. In Figure 5.3C, a promoter agent with a TF binding site is represented. Both the promoter and the operator agents are of "DnaRegion" type, and are identified with the `SO:0000167` and `SO:0000057` terms. Although the nucleotide information can be linked to existing repositories using the `bqbiol:is` term, for synthetic sequences agents can directly be annotated using SBOL terms. The term `sbol:nucleotides` is used to store the nucleotide sequences for these agents. A parent-child relationship between the promoter and the operator agents can be represented using an `sbol:SequenceAnnotation` RDF resource, which allows the location of an operator subpart to be specified.

This approach can be used to annotate a pattern with a specific entry from a database (patterns can also be stated as observables of the model). For instance, Figure 5.3D shows an example of such an observable. `Spo0A_p` represents the phosphorylated protein, which acts as a TF and is defined as an observable.

Figure 5.4 demonstrates annotation of rules. The first rule (Figure 5.4A) describes the binding of LacI TF to a promoter. This biological activity is described using the `GO:0008134` (*transcription factor binding*) term. In the second example (Figure 5.4B), a phosphorylation rule is annotated. The rule contains a subrule representing ATP to ADP conversion. This subrule is linked to the parent rule with the `hasSubrule` qualifier. Moreover, the annotation of the rate for this rule is presented in Figure 5.4C. The annotated Kappa

and BioNetGen models for a two-component system (TCS), controlling a simple promoter architecture can be found in directory[5].

Finally, in Figure 5.5 we present the fragment of a specific rule (taken from the TCS Kappa model) materialised using the `krdf` tool. The tool generates a version of the rules themselves in RDF together with the annotations (in this way the entire model is presented in a more uniform way).

5.6 Using Annotations

The discussed framework could be coupled to the development of tools that allow to extract and analyze the annotations embedded in a model. Several tools are currently under development. We demonstrate here the `krdf` tool that can be used for checking duplication of rules and inconsistencies between different parts of the model, basic problems encountered when composing and creating biological models (M L Blinov et al., 2008; Lister et al., 2009). Another application is to draw an annotated contact map visualising the entities involved, the interactions and the biological information stored in the annotations – this merges the classical notion of contact map used in Kappa models (Danos; Laneve, 2004; Danos; Jérôme Feret; W. Fontana; Russ Harmer, et al., 2009) with biological semantics.

The `krdf` tool operates on Kappa models and has several modes of operations that can provide increasingly more information about the model. The first, selected with the `-a` option, extracts the modeller's annotations. The second mode, selected with the `-m` option, *materialises* the information in the rules themselves into the RDF representation (as illustrated in Figure 5.5). Finally the `-n` option *normalises* the patterns present in the rules according to their declarations.

Once a complete uniform representation of the model in RDF has been generated, one can query it using SPARQL with a tool such as `roqet` (Beckett, 2015). For example, a SPARQL query can deduce a contact map—pairings of sites on agents that undergo binding and unbinding according to the rules in the model. These pairings form a graph that can be visualised using tools such as GraphViz (Ellson et al., 2002). With an appropriate query[6], `roqet` can output the result in a GraphViz format. A more sophisticated manipulation[7] can extract annotations from the RDF representation of the

[5] Files `tcs.kappa` and `tcs.bngl` in the `http://purl.org/rbm/rbmo/examples` directory respectively.

[6] See the `binding.sparql` file in the krdf directory.

[7] See the `contact.py` script in the krdf directory.

TCS example model and easily create a richly annotated contact map diagram (Figure 5.6). In this way, biological information extracted from the annotations can be added to the agents, sites and interactions (using GraphViz for rendering)[8].

An example of a simple version of such a query, to extract the binding interactions is below:

```
## Return a sequence of (rule, agentA, siteA, agentB, siteB) tuples
## that correspond to binding operations.

PREFIX rbmo: <http://purl.org/rbm/rbmo#>
SELECT DISTINCT ?rule ?agentA ?siteA ?agentB ?siteB
WHERE {
    ?rule rbmo:lhs [
        ## the left hand side of a rule has an agent
        ## with a site bound to nothing
        rbmo:agent ?agentA;
        rbmo:status [
            a rbmo:UnboundState;
            rbmo:isStatusOf ?siteA
        ]
    ]; rbmo:rhs [
        ## the right hand side of a rule has the same
        ## agent with the site bound to something.
        rbmo:agent ?agentA;
        rbmo:status [
            rbmo:isBoundBy ?binding;
            rbmo:isStatusOf ?siteA
        ]
    ] .

    ?rule rbmo:lhs [
        ## the left hand side of a rule has an agent
        ## with a site bound to nothing
        rbmo:agent ?agentB;
        rbmo:status [
            a rbmo:UnboundState;
            rbmo:isStatusOf ?siteB
        ]
    ]; rbmo:rhs [
        ## the right hand side of a rule has the same
        ## agent with the site bound to something.
        rbmo:agent ?agentB;
        rbmo:status [
            rbmo:isBoundBy ?binding;
            rbmo:isStatusOf ?siteB
        ]
```

[8]The tool assumes that only single instances of an agent are involved in a rule. It can be generalized.

```
    ] .
    ## this filter is necessary to check that the binding
    ## actually is one, that is blank nodes are used to
    ## bind sites
    FILTER isBlank(?binding)
    ## Apply a predictable ordering on the sites so that
    ## edges do not appear twice.
    FILTER (STR(?siteA) < STR(?siteB))
   ## ORDER BY ?rule
}
```

and together with the corresponding query for unbinding and some cosmetically motivated queries to extract human readable labels, this results in sufficient information to generate Figure reffig:contact.

Moreover, one can easily create a query that implements a join operation on the property of `bqbiol:is`, enforcing a stronger form of identity semantics than this predicate is usually given. A filter clause is necessary to prevent a comparison of a rule with itself (see the SPARQL query in Figure 5.7). In this way, the discussed annotations could also be used to detect duplication of rules (e.g, obtained when combining different biological models).

Another possible application of the presented annotation schema is the checking of inconsistencies in a rule-based model. This can be done in several different ways. A simple way is to use the replacement semantics of `owl:sameAs`. A statement of the form `a owl:sameAs b` means that every statement about `a` is also true if `a` is replaced by `b`. In particular if we have statements about the types of `a` and `b`, and these types are disjoint, the collection of statements is unsatisfiable (hence, the model has been found to be inconsistent). Then, an OWL reasoner such as HermiT (Shearer et al., 2008) or Pellet (Sirin et al., 2007) can derive that `a` and `b` have type `owl:Nothing`.

This can be implemented with the following work-flow (here only sketched): (i) generate the fully materialised RDF version of a model using, e.g. `krdf`. For each use of `bqbiol:is`, add a new statement using `owl:sameAs`; (ii) retrieve all ontologies that are used from the web. For each external vocabulary term with `bqbiol:is` or `bqbiol:isVersionOf` retrieve a description and any ontology that it uses (recursively). Merge all of these into a single graph. This graph contains the complete model and annotations, with entities linked using a strong form of equality to external vocabulary terms, and descriptions of the meaning of these vocabulary terms; (iii) the reasoner can be used to derive terms that

are equivalent to `owl:Nothing` and if any of these terms is found then an inconsistency has been identified. Using the proof generation facilities of OWL reasoners, the sequence of statements required to arrive to `foo rdf:type owl:Nothing` can be reproduced (in this way, the initial source of the inconsistency can be also identified).

5.7 Perspective and Future Works

In this chapter we have reviewed the recent proposal to incorporate annotations to rule-based models, following the approach recently presented in (Mısırlı; Cavaliere, et al., 2015). We have also discussed in a more general way the role of annotations and how they are strongly related to the notion of abstraction. In general, for consistency, we have followed the terms originally defined in (Mısırlı; Cavaliere, et al., 2015). However, the suggested standardized terms can be used in a complementary manner with existing metadata resources such as MIRIAM annotations and URIs, and existing controlled vocabularies and ontologies. Although, the approach has only described the annotations of Kappa and BioNetGen files, it can be easily applied to other rule-based models.

In particular, PySB (Lopez et al., 2013a) already includes a list of MIRIAM annotations at the model level, and can be extended to include the type of annotations described here. SBML's `multi`[9] package is being developed to standardise the exchange of rule-based models. The entities in this format inherit the annotation property from the standard SBML and can therefore include RDF annotations. These SBML models could thus be imported or exported by tools such as KaSim or RuleBender, avoiding the loss of any biological information.

It is important to remark that annotations are also useful for automated conversions between different formats. Conversion between rules and reaction networks is already an ongoing research subject (M L Blinov et al., 2008), and the availability of annotations can play an important role for reliable conversion and fine-tuning of models (Tapia et al., 2013; L. A. Harris et al., 2015). It is straightforward to use the framework presented and automatically map agents and rules to glyphs (Chylek; Hu, et al., 2011) or to convert models into other visual formats such as SBGN or genetic circuit diagrams (Mısırlı; J. S. Hallinan, et al., 2011).

More generally, annotations are designed for machine readability and can be produced computationally (e.g., by model repositories). This can be done by developing APIs and tools to access to a set of biological parts (Cooling et al., 2010; Mısırlı; J.

[9] `http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/multi`

Hallinan, et al., 2014) that will incorporate rule-based descriptions and will be annotated with the proposed schema (the authors are currently working on this line of research). This will open the possibility of composing together rule-based models extracted from distinct repositories. Tools such as Saint (Lister et al., 2009) and SyBIL (Michael L Blinov; Ruebenacker, et al., 2010) could be extended to automate the annotation of rule-based models and help to address the automatic annotation of models. In this way, the extensive information available in biological databases and the literature can be integrated and made available via rule-based models, taking advantage of the syntax and the framework presented in this chapter.

One of the ultimate goals is to use annotations for helping the automatic composition of rule-based models. As recently suggested in (Mısırlı; Waites, et al., 2016) the proposed schema can be used to automate the design of biological systems using rule-based model with a workflow that combines the definition of modular templates to instantiate rules for basic biological parts. The templates, defining rule-based models of basic biological parts[10] can be associated with quantitative parameters to create particular parts models, which can then be merged into executable models. Such models are annotated using the reviewed schema leading to a feasible protocol to automate their composition for the scalable modelling of synthetic systems (Mısırlı; Waites, et al., 2016).

The described annotation ontology for rule-based models can be found at `http://purl.org/rbm/rbmo` while the tool and all the presented examples can be found at `http://purl.org/rbm/rbmo/krdf`.

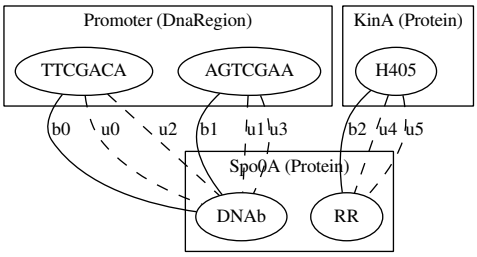
[10] Available at `http://github.com/rbm/composition`

```
A:
#^:LacI.pLac a rbmo:Rule  ;
#^  bqbiol:isVersionOf go:GO:0008134 ;
#^  dct:title "Dna binding" ;
#^  dct:description "TF1 binds to the promoter" .
'LacI.pLac' Target(x~p), Promoter(tfbs1,tfbs2) <-> Target(x~p!1), Promoter(
    tfbs1!1,tfbs2) @kf,kr
B:
#^:S_phosphorylation a rbmo:Rule ;
#^   bqbiol:isVersionOf sbo:SBO:0000216 ;
#^   dct:title "S Phosphorylation" ;
#^   dct:description "S is phosphorylated" ;
#^   rbmo:hasSubrule :S_phosphorylation:1 .
#^:S_phosphorylation:1 a rbmo:Rule ;
#^   bqbiol:isVersionOf sbo:SBO:0000216 ;
#^   dct:title "ATP -> ADP" ;
#^   dct:description "ATP to ADP conversion" .
'S_phosphorylation' S(x~u!1), K(y!1) | 0.1:ATP -> S(x~p), K(y) | 0.1:ADP
    @kp
C:
#^:kp a sbo:SBO:0000002 ;
#^  bqbiol:isVersionOf sbo:SBO:0000067 ;
#^  dct:title "Phosphorylation rate" .
```

Figure 5.4: Annotating rules and variables. **A.** TF DNA binding rule. **B.** Phosphorylation rule with a subrule for the ATP to ADP conversion. **C.** Annotation of a phosphorylation rate variable.

```
:As1As2Spo0A_to_As2Spo0A a rbmo:Rule ;
  dct:title "Cooperative unbinding" ;
  rbmo:lhs [
    a rbmo:Pattern ;
    rbmo:agent :Spo0A ;
    rbmo:status [
      rbmo:isBoundBy :As1As2Spo0A_to_As2Spo0A:left:1 ;
      rbmo:isStatusOf :Spo0A:DNAb ;
      a rbmo:BoundState ;
    ], [
      rbmo:internalState :Spo0A:RR:p ;
      rbmo:isStatusOf :Spo0A:RR ;
      a rbmo:UnboundState ;
    ] ;
  ].
```

Figure 5.5: Fragment of the RDF representation of a materialised rule obtained by merging the metadata supplied by the model author with an RDF representation of the rule. The left hand side of the rule contains a pattern involving `:Spo0A` and that there are two pieces of state information: The first one refers to the `:Spo0A:DNAb` site, and it is bound to something (that can only be recovered using the rest of the model, not presented here). The second refers to the `:Spo0A:RR` site, it has a particular internal state, and it is unbound.



b0: Spo0A binding to Operator 1
b1: Spo0A binding to Operator 2
b2: Spo0A-KinA binding
u0: Cooperative unbinding: Spo0A unbinds from Operator 1
u1: Cooperative unbinding: Spo0A unbinds from Operator 2
u2: Spo0A unbinding from Operator 1
u3: Spo0A unbinding from Operator 2
u4: Spo0A(phosp)-KinA unbinding
u5: Spo0A(unphos)-KinA unbinding

Figure 5.6: Contact map generated by a SPARQL query on the RDF materialisation of the TCS example in Kappa. Biological information concerning the agents, rules and sites, types of the molecules, DNA sequences and typology of the interaction, are extracted automatically from the model annotations. Figure has been redrawn as in (Mısırlı; Cavaliere, et al., 2015).

```
SELECT DISTINCT ?modelA ?ruleA ?modelB ?ruleB
WHERE {
  ?ruleA a rbmo:Rule;
      dct:isPartOf ?modelA;
      bqbiol:is ?ident.
  ?ruleB a rbmo:Rule;
      dct:isPartOf ?modelB;
      bqbiol:is ?ident.
  FILTER (?ruleA != ?ruleB)
}
```

Figure 5.7: Detection of duplicate rules.

Chapter 6

A Genetic Circuit Compiler

Generating Combinatorial Genetic Circuits with Web Semantics and Inference

Abstract. A central strategy of synthetic biology is to understand the basic processes of living creatures through engineering organisms using the same building blocks. Biological machines described in terms of parts can be studied by computer simulation in any of several languages or robotically assembled *in vitro*. In this paper we present a language, the Genetic Circuit Description Language (GCDL) and a compiler, the Genetic Circuit Compiler (GCC). This language describes genetic circuits at a level of granularity appropriate both for automated assembly in the laboratory and deriving simulation code. The GCDL follows Semantic Web practice and the compiler makes novel use of the logical inference facilities that are therefore available. We present the GCDL and compiler structure as a study of a tool for generating κ-language simulations from semantic descriptions of genetic circuits.

6.1 Introduction

Synthetic biology extends classical genetic engineering with concepts of modularity, standardisation, and abstraction drawn largely from computer engineering. The goal

The work presented in Chapter 6 was previously published in the journal *ACS Synthetic Biology* as "A Genetic Circuit Compiler: Generating Combinatorial Genetic Circuits with Web Semantics and Inference" by William Waites, Göksel Mısırlı, Matteo Cavaliere, Vincent Danos and Anil Wipat (2018). The work was conceived by all of the authors. I originated the concept of using inference to materialise implicit information to achieve the goal of succinctness, designed, implemented and tested the `kcomp` compiler software and templates, produced the figures (except where otherwise noted) and drafted the majority of the text.

is ambitious: to design complex biological systems, perhaps entire genomes, from first principles (Baldwin, 2012). This enterprise has met with some success such as microbial drug synthesis (Paddon et al., 2013; Galanie et al., 2015), production of new biofuels (Ferry et al., 2012), and alternative approaches to disease treatment (Ruder et al., 2011). However, most applications are still small and mostly designed manually.

The are several obstacles to designing more complex circuits. The design space of potential circuits is very large. Even when a design is chosen, there is large *a priori* uncertainty about what its behaviour will be. In many cases the available information about molecular interactions in a cell is incomplete. A secondary obstacle is that designs can be brittle and very sensitive to the host environment in which they execute. In this context computational techniques become important for identifying biologically feasible solutions to problems of biological system synthesis. Beyond the challenges of the huge design space and associated uncertainties, writing these programs by hand is time-consuming and error prone, and there are very few tools available for verification and debugging them. Descriptions of models in terms of simulation code are tightly coupled to the language of the simulation program, and it may be difficult or impossible to use a different interpreter without completely rewriting the code.

We solve these problems by providing a high-level, modular, implementation-independent language for describing gene circuits called the Genetic Circuit Description Language (GCDL) and a compiler called Genetic Circuit Compiler (GCC). We use a strategy of contextual reasoning to obtain flexible output from this succinct input, and *templates* to support any number of output languages and modelling granularities. An overview of information flow through the compiler is shown in Figure 6.1. We demonstrate the utility of this approach by describing, compiling and simulating a complete genetic circuit, the well-known Elowitz repressilator (Elowitz et al., 2000). The compiler and example code are available at `https://github.com/rulebased/composition`.

Code generation from this high-level description to a low-level language for simulation greatly reduces the scope for error in coding simulations. Because the language is implementation-independent, it is not tightly coupled to any particular interpreter or hardware. In this way GCDL facilitates *evergreen models*, models that are specified sufficiently well to be unambiguous but not so specifically that they can only be executed or constructed in one software package or environment.

Domain specific languages and examples of compilers processing these languages have previously been shown (Pedersen et al., 2009; Beal et al., 2011; Cai; Beal, et al.,

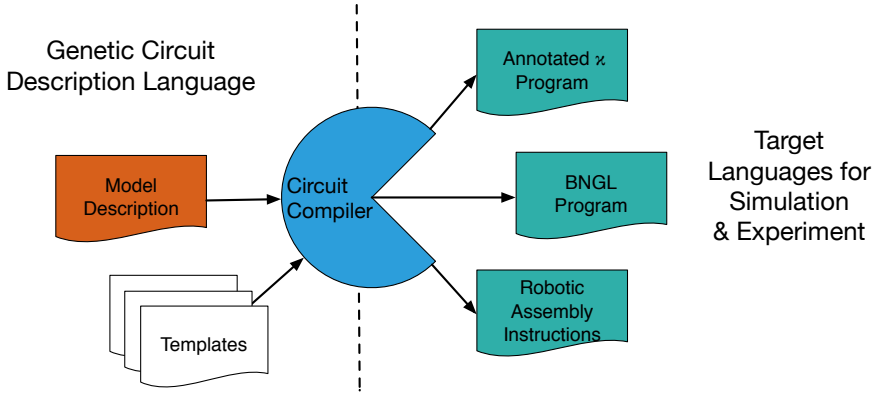


Figure 6.1: High-level data flow through the compiler. The compiler for synthetic gene circuits takes a model description written in GCDL and, using language-appropriate appropriate templates, creates code for simulation and laboratory assembly. We have implemented templates for annotated-κ for the KaSim software, and envision similar for the BNGL as well as SBOL.

2011; J. Hallinan et al., 2014). These languages are designed to allow for simulations using a particular methodology such as solving systems of ordinary differential equations or using Monte Carlo simulations. Unlike previous approaches, we emphasise the use of abstraction to facilitate *retargeting* or production of output suitable for different simulation environments and techniques as well as automated circuit assembly in the laboratory from a single description. Compiler targets are implemented using conditional inference, defining the semantics of the terms used in the description of the circuit in a way that is determined by the desired output type. The design of the compiler is general, and not limited to the present context of genetic circuits. The design is shown schematically in Figure 6.2.

The GCDL is an RDF (Cyganiak et al., 2014) vocabulary and attendant inference rules which facilitates gathering and collation of information about the constituent parts of a genetic circuit (Neal et al., 2014). The output programs can be specialised to various languages, such as the KaSim flavour of κ (Danos; Jérôme Feret; W. Fontana; Russell Harmer, et al., 2007; Krivine et al., 2018), BioNetGen's BNGL (Michael L Blinov; James R Faeder, et al., 2004; Leonard A. Harris et al., 2016), other representations such as SBOL (Galdzicki; Clancy; Oberortner; Pocock; J. Y. Quinn, et al., 2014) or indeed whichever form is required by robotic laboratory equipment that assembles circuits *in vitro*. This output flexibility is accomplished using *templates* that use facts derived by inference rules (Berners-Lee, 2005) from the input model.

We now proceed as follows. In Section 6.2 we give an overview of those aspects of

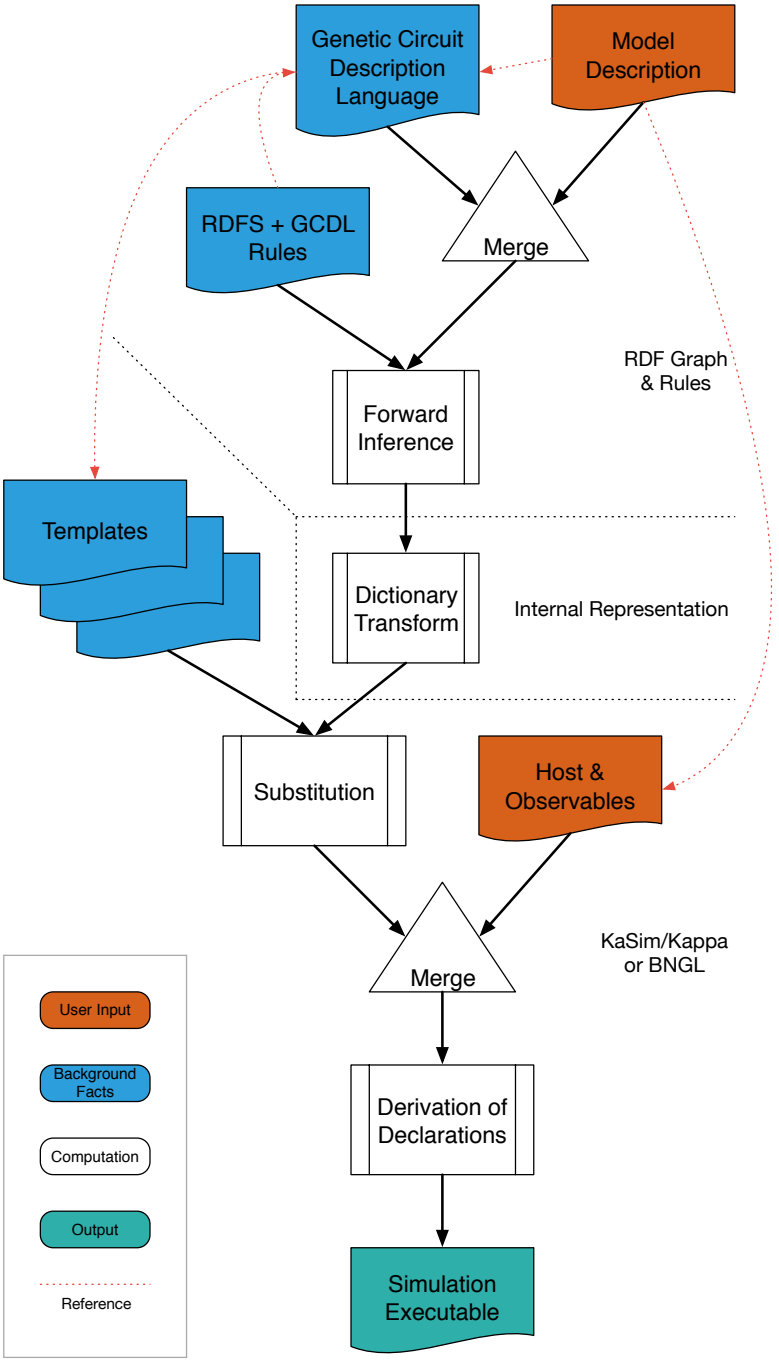


Figure 6.2: Detailed data flow through the compiler. This illustrates the use of inference to expand the GCDL model to derive consequent information appropriate to producing the next stage of output in the specific target language.

synthetic biology and genetic engineering that are necessary to contextualise our work. Next, in Section 6.3, we explain the representation of this kind of genetic circuit model in GCDL, this is the main input to the compiler. In order to understand the desired output of the compiler, in Section 6.4 we illustrate how these constructs are represented as rule-based code for the κ language simulator, KaSim. There follows a discussion in Section 6.5 of how the compiler infers the executable model from the input description. Finally, in Section 6.6 we discuss some possible uses and limitations of our technique.

6.2 Background

6.2.1 Rule-based Modelling of Genetic Processes

A weakness of reaction-based methods for modelling the processes of transcription, translation and the production of chains of proteins is that they require chemical species for each bound state of the reagents. This in turn requires specification of reactions for each combination of these reagents. To solve this problem of needing combinatorially many reactions to describe substantially the same process, a generalisation of reactions called *rules* are used (Hlavacek et al., 2003; Danos; Laneve, 2004; Danos; Jérôme Feret, et al., 2008).

In the rule-based representation, *agents* correspond to reagents and they can have slots or *sites* that can be bound, or not. They can also have internal state. Unlike reactions which have no preconditions apart from the presence of the reagents, with rules, a configuration of the sites—bound in a particular way, bound in some way, unbound, or unspecified—is a precondition for the application of the rule. A rule may re-arrange the bonds, creating or destroying them, without the need to invent new agents in order to represent different configurations of a given set of molecules.

The reader should note that the word *rule* is used in two distinct senses in this chapter. The first is as we have just described. The second is in the sense of *inference rule* as used in logic and in particular the way in which we deduce executable rule-based models from their declarative representations in RDF.

6.2.2 The κ Language

To briefly illustrate the essentials of rule-based modelling we will use the language of the Kappa simulation software, KaSim (Krivine et al., 2018). An agent declaration and rule expressing the formation of a polymer can be written as,

```
%agent: A(d,u)

'binding' A(u[.]), A(d[.]) -> A(u[1]), A(d[1]) @k
```

We can gloss this as an agent with two sites, `u` and `d` for upstream and downstream, and a rule. The rule concerns two agent patterns one of which has an unbound upstream site, and the other an unbound downstream site, and the action of the rule is to bind them, the notation `[1]` denoting the bond. This process happens at some rate `k`.

The state of the other site of each agent is left unspecified, so implicit in this rule is the possibility that either or both the agents may already be bound to others and so part of arbitrarily long chains. In other words this expression covers not only two monomers joining together but an n-mer and an m-mer for arbitrary n and m. This is the essence of the expressive advantage that rule-based modelling provides. To express a similar concept using a reaction network would in fact require infinitely many reagents for every possible n (and m) and infinitely many reactions for every possible combination.

6.2.3 Biological Parts and Annotation

For efficiency, and economy of representation, we claim that the description of a computational model should include minimum information necessary for simulation. However, in order to use these models in an automated design process, additional metadata, or *annotations*, about the meaning of different modelling entities is needed (Neal et al., 2014). Annotation facilitates the drawing of specific parts from a database such as the Virtual Parts Repository (Mısırlı; J. Hallinan, et al., 2014). Models in that database are annotated with machine-readable metadata intended for combination into larger models. Myers and his colleagues have used annotations to derive simulatable models from descriptions of genetic circuits (Roehner et al., 2015) and vice versa (Nguyen et al., 2016), though these use reaction-based techniques and so inherit the poor scaling properties of that method.

To facilitate the *in silico* evaluation of potential synthetic gene circuits, a library of descriptions of genetic parts, together with their modular models is suggested in (Cooling et al., 2010; Mısırlı; J. Hallinan, et al., 2014). These parts are intended to be large enough to have a particular meaning or function (i.e. larger than individual base pairs) but not so large that they lack the flexibility to be recombined (i.e. entire genes). Thus we are concerned with coding sequences for particular proteins, promoters that, when activated, start the transcription process, operators that activate or suppress promoters

according to whether they are bound or not by a given protein, and a small number of other objects. A sequence of these objects is a genetic circuit, and our goal is to have a good language for describing such sequences.

Annotation in this setting means machine-readable descriptions of entities of biological interest. This is done with statements, triples of the form (subject, predicate, object) according Semantic Web standards (Cyganiak et al., 2014; Harmelen et al., 2004). Entities are identified with Universal Resource Identifiers (URIs) (Masinter et al., 2005). This provides the dual benefit of globally unique identifiers for entities and a built-in mechanism for retrieving more information about them providing that some care is taken to publish data according to best practises (Hyland et al., 2014; Sauermann et al., 2011). Large bodies of such information about biologically relevant information are published on the Web (M. Ashburner et al., 2000; U. Consortium et al., 2008) and the use of Semantic Web standards for annotating our models allows us to express how an entity in a model description corresponds to a real world protein, or gene sequence or other entity.

The Semantic Web also affords us a technical advantage: inference rules. These can be either explicit as in Notation3 (Berners-Lee; Connolly, et al., 2008; Berners-Lee; Connolly, 2011) or implicit as in OWL Description Logics (Horrocks, 2005; Brickley et al., 2014). In either case this facility makes it possible, given a set of statements, to derive new statements according to inference rules. We use this to improve the ergonomics of our high-level language: while the compiler itself will make use, internally, of a large amount of information, we do not expect the user to supply it in painstaking detail. Rather, we allow the user to specify the minimum possible and provide rules to derive the necessary detail. Inference rules provide for both economy of representation for the high-level model description and flexibility for the different implementations.

6.3 A Language for Synthetic Gene Circuits

This section describes the GCDL, the high-level language for describing genetic circuits made from standard biological parts (Cooling et al., 2010)(Mısırlı; J. Hallinan, et al., 2014). We begin by stating the properties that we want in such a language and showing how we achieve them. There follows a synopsis of the vocabulary terms essential to the language. Finally, we illustrate salient language features applied to example circuits.

6.3.1 Desired Language Features

Our desired language features for high-level representation of a genetic circuit are as follows,

1. sufficiency, there should be enough information to derive executable code for the circuit,
2. identifiability, it should be possible to determine to which biological entities (DNA sequences, proteins) the representation refers,
3. extensibility, it should be straightforward to add information or constructs that are not presently foreseen,
4. generality, there should be no requirement that information about biological parts comes from any particular set or source, and
5. concision, there should be a minimum of extraneous detail or syntax.

The third and fourth requirements are readily met by using RDF as the underlying data model. The *open world* presumption (Drummond et al., 2006) means that adding information as necessary is straightforward. The use of URIs (Masinter et al., 2005) which can be dereferenced to obtain the required information means that information from different web-accessible databases can be obtained, mixed and matched as desired. The use of URIs goes some way towards meeting the second requirement, albeit with some well-known caveats (Halpin et al., 2010).

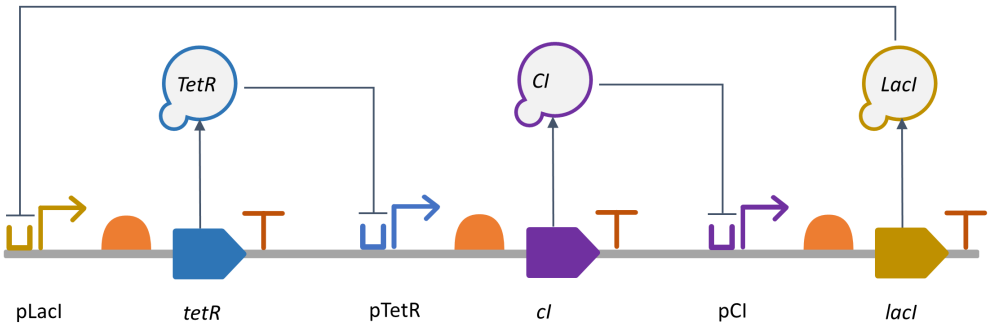
The first and last of the desired features are the primary areas of innovation of the present work. We suggest (but do not require) the use of Turtle (Prud'hommeaux; Carothers, 2014) or indeed Notation3 (Berners-Lee, 2005) as the concrete surface syntax for writing models. This goes some way towards a representation that is intelligible by humans. Even then, we aim to minimise what needs to be written and we do this using inference rules—if a needed fact can be derived from the model under the provided rule-set, it is unnecessary to write it explicitly in the model. Indeed it may even be undesirable to do so since it is a possible source of errors, for example some kinds of assertions may be correct in the context of some output types and incorrect in others. We aim for a minimal, yet complete under the inference rules, description of the model.

6.3.2 Vocabulary Terms

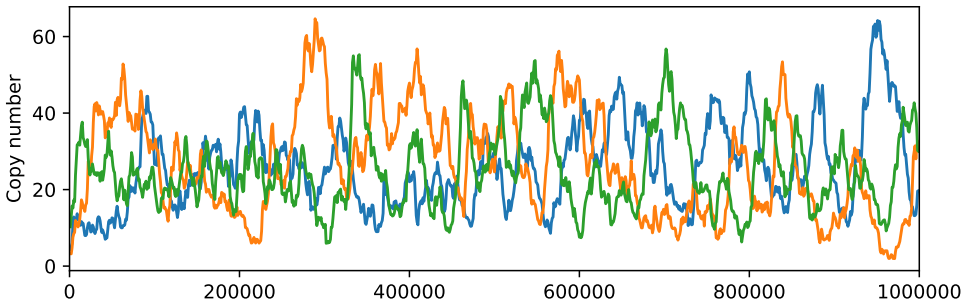
New terms introduced in this paper have the prefix `gcc` which can be read as the "Genetic Circuit Compiler" vocabulary. The list of terms is reproduced in Table 6.3 and their complete definitions are given in Appendix B.1 together with the accompanying rules

in Appendix B.2. The GCDL is the union of terms from the *gcc* namespace with those from the Rule-Based Model Ontology (RBMO) that we previously defined (Mısırlı; Cavaliere, et al., 2015) together with terms from the Simple Knowledge Organization System (SKOS) (Miles et al., 2005) vocabulary, RDF Schema (RDFS) (Brickley et al., 2014) and Resource Description Framework (RDF) (Cyganiak et al., 2014).

6.3.3 Model Description



(a) An example genetic circuit: the Elowitz repressilator. It is a negative feedback oscillator. The circuit is arranged linearly. Protein production and inhibitory protein-operator relationships are shown using the SBOL visual standard. (Figure drawn by G. Mısırlı)



(b) Sample simulation data from a program produced by the compiler showing the expected oscillations. Note in particular the relatively small copy numbers of the proteins for which stochastic simulation in the κ language is well suited.

Figure 6.3: Diagram and sample simulation results of the Elowitz repressilator

To illustrate the syntax of the high-level language, we use the well known Elowitz repressilator shown diagrammatically in Figure 6.3a. The complete model can be found in Appendix B.3 as well as distributed in the `examples/` subdirectory of the compiler distribution. Also included with the compiler is a hand-assembled implementation of this circuit for comparison. A sample trace produced by generated program is shown in

Classes	
gcc:Part	Generic biological part
gcc:Operator	Operator
gcc:Promoter	Promoter
gcc:RibosomeBindingSite	Ribosome Binding Site
gcc:CodingSequence	Coding Sequence
gcc:Terminator	Terminator
gcc:Token	Token or symbol in a template

Predicates	
gcc:include	Include a low-level model fragment
gcc:prefix	The prefix to use for generated annotations
gcc:init	Specifies initial copy numbers
gcc:part	Links a part to its token or symbol
gcc:overlaps	Indicates that two parts overlap (symmetric)
gcc:linear	Linear circuit type
gcc:circular	Circular circuit type
gcc:transcriptionFactor	Relates an operator to its transcription factor
gcc:transcriptionFactorBindingRate	Various rates
gcc:transcriptionFactorUnbindingRate	
gcc:rnapBindingRate	
gcc:rnapUnbindingRate	
gcc:rnapRNAUnbindingRate	
gcc:ribosomeBindingRate	
gcc:ribosomeRNAUnbindingRate	
gcc:ribosomeProteinUnbindingRate	
gcc:transcriptionInitiationRate	
gcc:transcriptionElongationRate	
gcc:translationElongationRate	
gcc:rnaDegradationRate	
gcc:proteinDegradationRate	

Table 6.3: Selected terms from the GCC vocabulary

```
## Model declaration
:m a rbmo:Model;
  ## bibliographic metadata
  dct:title "The Elowitz repressilator constructed from BioBrick parts";
  dct:description "Representation of the Elowitz repressilator given in
  the Kappa BioBricks Framework book chapter";
  rdfs:seeAlso <http://link.springer.com/protocol/10.1007/978
  -1-4939-1878-2_6>;
  gcc:prefix <http://id.inf.ed.ac.uk/rbm/examples/repressilator#>;
  ## include the host environment
  gcc:include <.../host.ka>;
  ## initialisations
  gcc:init
    [ rbmo:agent :RNAp; gcc:value 700 ],
    [ rbmo:agent :Ribosome; gcc:value 1000 ];
  ## The circuit itself, a list of parts
  gcc:linear (
    :R0040o :R0040p :B0034a :C0051 :B0011a
    :R0051o :R0051p :B0034b :C0012 :B0011b
    :R0010o :R0010p :B0034c :C0040 :B0011c
  ).

```

Figure 6.4: Example model for a synthetic gene circuit, Elowitz' repressilator.

Figure 6.3b. Figure 6.4 shows a description of this the core of the model, in the GCDL. Some bibliographic metadata is included, using the standard Dublin Core (Kunze et al., 2007) vocabulary, as well as a generic pointer (`rdfs:seeAlso`) to a publication about this model.

The term `gcc:prefix` is necessary in every model, it instructs the compiler that any entities that it creates should be created under the given prefix. Ultimately annotated rules will be generated for the low-level representation and the annotated entities require names. To give them names, a namespace is required and this is how it is provided.

Next there is a `gcc:include` statement. This is a facility for including extra information in the low-level language. Extra information typically means rules for protein-protein interactions which are beyond the scope of the current work and as such it is simply supplied as a program fragment in the output language. This corresponds roughly to calling an assembly or machine language routine to perform a specialised task when programming a computer in a high-level language like C.

There follows initialisation for specific variables. In this case these are the copy

```
:C0012 a gcc:CodingSequence;
    gcc:label "Coding sequence for LacI";
    gcc:part "C0012";
    gcc:protein :P0010;
    gcc:proteinDegradationRate 0.0001.

:P0010 a gcc:Protein;
    bqbiol:is uniprot:P03023;
    skos:prefLabel "P0010";
    rdfs:label "LacI".

```

Figure 6.5: A coding sequence part description from the repressilator model. Notice how the coding sequence is linked to the protein that it codes for.

numbers for RNA polymerase molecules and ribosomes. These are denoted using `rbmo:agent` because of our choice to support rule-based modelling for greater generality than reaction-based methods. Finally, the circuit itself is specified. The argument, or object is an `rdf:List` which simply contains identifiers for the parts, in order.

The circuit itself is now defined. However at this juncture, we simply have a list of parts without having specified what they are or what their intended behaviour is. To obtain a working model, we need more.

6.3.4 A Part Description

A simple example of a part description is shown in Figure 6.5. This is a coding sequence, as is clear from the type annotation on the part. It codes for a particular protein, specified with `gcc:protein`. This term is specific to proteins because under normal circumstances other kinds of part do not code for proteins. It is given a part symbol using `gcc:part` because the output language will not typically permit the use of URIs as identifiers, so this symbol via the implied `skos:prefLabel` (Miles et al., 2005) is what will appear instead. The protein produced by this coding sequence is also specified and linked using `gcc:protein`. It too is given a label using `skos:prefLabel` for the same reason, and its degradation rate is also specified with `gcc:proteinDegradationRate`. It is equally possible to specify the rates for transcription and translation in a similar manner though not shown here. In practice, rates are known primarily from experiment and this is an important reason to have accessible databases or repositories of part specifications.

Importantly, following the practice in our previous paper on rule annotation (Mısırlı; Cavaliere, et al., 2015), a weak identity assertion is made with identifiers in external databases for the parts. This uses `bqbiol:is` instead of `owl:sameAs` because the strong replacement semantics (Leibniz' Law (Forrest, 2016)) of the latter can yield unwanted inferences when terms are not used perfectly rigorously (Halpin et al., 2010). This weaker identify assertion permits the identification of the `:P0010` in the example with the identifier for the protein in the well-known UniProt (U. Consortium et al., 2008) database.

6.3.5 A More Complex Part Description

A more involved example demonstrating how an operator-promoter combination is encoded is shown in Figure 6.6. Here we have an operator with the rates for binding and unbinding of the transcription factor specified explicitly. If the operator is bound by the transcription factor, the neighbouring promoter is repressed—an RNA polymerase will not be able to bind. By contrast if the operator is unbound, the promoter will accept binding of RNA polymerase easily and frequently. The language supports an arbitrary amount of operator context for operators and promoters enabling the specification of complex regulatory structures such as combinatorial logic gates (R. S. Cox et al., 2007; Wang et al., 2011; Sanchez et al., 2011) and some forms of cooperative binding.

The transcription factor is specified by using `gcc:transcriptionFactor` to refer to the protein that will turn the operator on or off. Like `gcc:protein` for coding sequences, the term is unique to operators.

The promoter comes next and it is the most complex part to specify. Because the rate for binding of RNA polymerase depends on the state of the operator, two rates must be specified. States of the nearby parts are specified using the `rbmo` vocabulary which makes available the full range of expressiveness for rule-based output languages. For generality, a list of parts, upstream or downstream on the DNA strand may be specified along with their states. This enables a promoter to be controlled by two or more operators. The rate itself in this case is given with `gcc:value` for each case.

6.3.6 Host and Protein-Protein Interactions

The language can also support protein–protein interactions in a basic way. To see why these are useful, consider an example from the engineering of a bacterial communication system where the subtilin molecule is used to control population level dynamics. Cells

```
:R0040o a gcc:Operator;
  rdfs:label "TetR activated operator";
  gcc:part "R0040o";
  gcc:transcriptionFactor :P0040;
  gcc:transcriptionFactorBindingRate 0.01;
  gcc:transcriptionFactorUnbindingRate 0.01.

:R0040p a gcc:Promoter;
  rdfs:label "TetR repressible promoter";
  gcc:part "R0040p";
  gcc:rnapBindingRate
    [
      gcc:upstream ([ a rbmo:BoundState;
                      rbmo:stateOf :R0040o ]);
      gcc:value 7e-7
    ], [
      gcc:upstream ([ a rbmo:UnboundState;
                      rbmo:stateOf :R0040o ]);
      gcc:value 0.0007
    ].

```

Figure 6.6: An operator and promoter from the repressilator model. The binding rates for the promoter depend on the state of the adjacent operator.

have the receiver device (Bongers et al., 2005; Mısırlı; J. Hallinan, et al., 2014) to sense the existence of subtilin, and the reporter device to initiate downstream cellular processes (Figures 6.7a and 6.7b). In the subtilin receiver, the interactions among the proteins produced by translation and the operator-promoters are mediated by a cascade reaction initiated by the subtilin molecule. Subtilin combines to phosphorylate the *SpaK* protein, which in turn phosphorylates the *SpaR* protein that finally binds to the promoter that controls the emission of a fluorescent green protein.

While the genetic circuit can straightforwardly be described similarly to the previous repressilator example, the protein–protein interactions cannot. We do not attempt here to model these interactions in the GCDL though a future extension could do so. Instead we simply allow for inclusion of the relevant program, as a file in the output language (in this case κ-language). It is possible to supply arbitrary code in the low-level language using the `gcc:include` term. This facility makes it feasible to represent such genetic circuits which depend strongly on the host environment in order to operate.

6.3.7 Protein Fusion

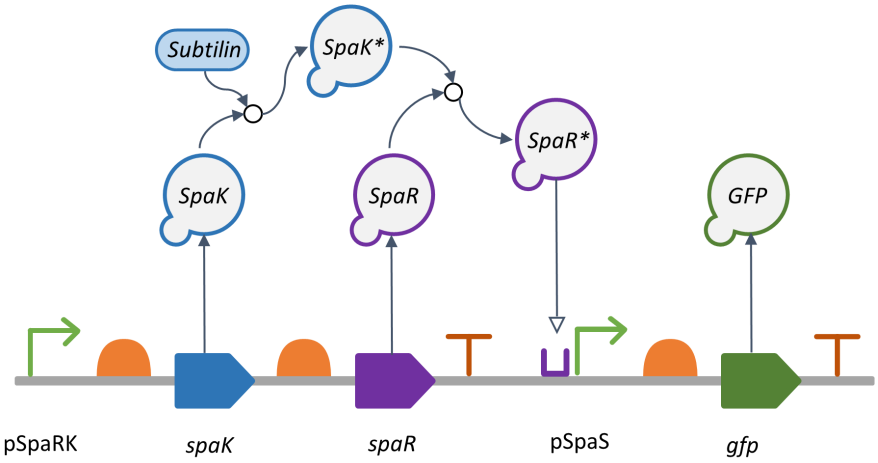
It is also worth noting that this example illustrates that in the high-level language it is immediately possible to represent devices that produce chains of proteins. This is known as protein fusion and is interesting for some applications (K. Yu et al., 2015). A chain of proteins is produced by adding adjacent (and appropriate) coding sequences. It is enough to simply list the coding sequences in the circuit; nothing else need be done.

6.3.8 Other Parts

The descriptions for the other kinds of biological parts, terminators, coding sequences, follow a similar pattern. There are terms for specifying the rates for the rules in which they participate, and a few specialised terms according to the function of the specific part. It is possible to find the available terms out by inspecting the `gcc` vocabulary included in Appendix B.1.

6.4 Output Representation

We now briefly consider the form of the output representation. By using different templates, the compiler can produce output in different languages. We focus on rule-based representations here and use the language of the KaSim simulator (Krivine et al., 2018)



(a) Diagram of the subtilin genetic circuit. The figure shows the multirelay phosphorylation, and hence the activation, of SpaR TFs to induce the downstream gene expression. As a result, GFP reporter proteins are produced in the presence of Subtilin molecules. (Figure drawn by G. Mısırlı)

```
:m a rbmo:Model;
    dct:title "Subtilin Receiver Two-Component System";
    gcc:include <.../subtilin-host.ka>;
    gcc:linear (
        :pSpaRK :RBSa :spaK :RBSb :spaR :Ta
        :pSpaS  :RBSc :gfp :Tb
    ).

```

(b) Corresponding semantic model.

Figure 6.7: Representations of the Subtilin Receiver model.

for concrete illustration as it is widely adopted for stochastic simulation of rule-based models (Wilson-Kanamori et al., 2015). The rule-based modelling approach is merely outlined here and follows that used in Kappa BioBricks Framework (KBBF) (Wilson-Kanamori et al., 2015) closely. We stress that though output as executable program in the KaSim language is demonstrated here, alternative rule-based representations like BioNetGen are equally possible as are descriptions in a language like SBOL as input to an experimental process in the laboratory.

The remainder of this section differs from the published version of this chapter in its detail. In the published version, for brevity, only the first sliding rule is included and explained. The remaining rules were relegated to supplementary materials. Here, we restore the section.

```
%agent: D-LacI(us, ds, bs)
%agent: R-LacI(us, ds, bs)
%agent: P-LacI(us, ds, bs)

```

(a) Distinct agents for each variant of part.

```
%agent: DNA(us, ds, bs, type{LacI})
%agent: RNA(us, ds, bs, type{LacI})
%agent: Protein(us, ds, bs, type{LacI})

```

(b) Generic agents for each variant with part indicated by the `type` site.

Figure 6.8: Dual representations of parts as agents.

6.4.1 Generic Agents

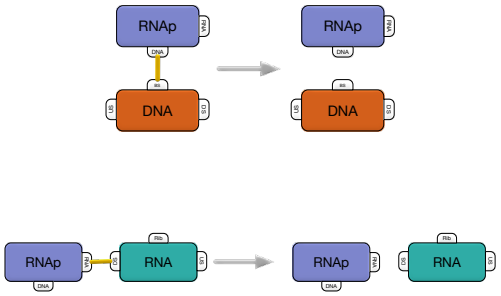
The behaviour of each kind of genetic part can be specified with rules, examples of which are given below. Fundamentally these rules operate on representations of DNA, RNA and proteins. Since each part can be linearly adjacent to others, there must be sites to stand for this linkage. These will be called `us` and `ds` for "upstream" and "downstream" respectively. There is also a need for a site to stand for the binding of protein or RNA polymerase to DNA, or the ribosome to RNA. This will be called `bs` for "binding site".

We immediately arrive at a modelling choice: the specific part, for example an operator to which the Lac repressor binds, could be represented as distinct kind of agent with DNA, RNA and protein variants (Figure 6.8a) or it could be represented as a label or tag on a generic DNA, RNA and protein agents (Figure 6.8b). We choose the latter because not only does it remove the need for having a large number of agents and inventing names for each DNA and RNA variant, but it greatly simplifies the rules. As we shall see the generic representation means that rules can easily be written where it only matters that a part is adjacent to *some* other part without specifying which one in particular. This is simply done by not specifying the `type` site. This is not possible with distinct agents because the Kappa language does not allow for unspecified or wild card agents.

These constructs, with their upstream and downstream linkages are enough to form the "rails" along which transcription and translation happen but we still require agents to join these together, namely RNA polymerase and the ribosome. These agents have

```
%agent: RNAp(dna, rna)
%agent: Ribosome(rna, protein)

```

Figure 6.9: RNA polymerase and ribosome agents.




```
'transcription-termination' DNA(bs[1]), RNAp(dna[1])
                          -> DNA(bs[.]), RNAp(dna[.]) @k

'translation-termination' RNA(ds[1]), RNAp(rna[1])
                          -> RNA(ds[.]), RNAp(rna[.]) @k
```

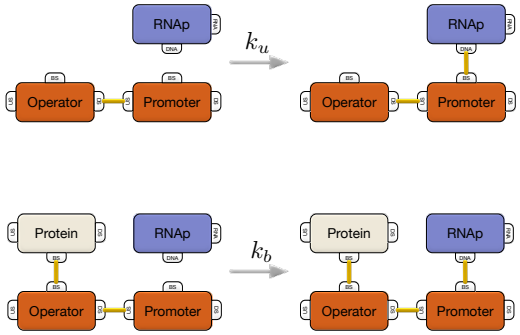
Figure 6.10: Termination rules: transcription and translation

two sites, one for each rail that they straddle (Figure 6.9).

6.4.2 Unbinding Rules

To understand how this works in practice, consider the simplest kind of rule, the unbinding rule. Those for transcription and translation are shown in Figure 6.10. This does not yet use any of the features that motivated our choice of agent representation, but does already show the "don't care, don't write" way of the KaSim dialect of Kappa: those sites that are not necessary for the operation of the rule do not appear. This brevity is a great boon.

An unbinding rule of the same form exists for each DNA part. Particularly significant among these is the unbinding of a protein from an operator.



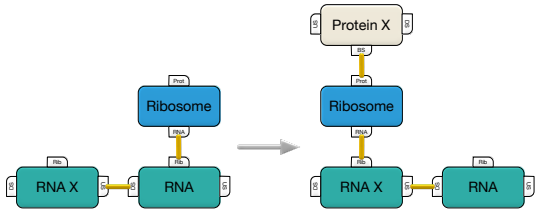

```
'RNAp-binding-unbound' DNA(type{operator}, ds[1], bs[.]),
                       DNA(type{promoter}, us[1], bs[.]),
                       RNAp(dna)
                    -> DNA(type{operator}, ds[1], bs[.]),
                       DNA(type{promoter}, us[1], bs[2]),
                       RNAp(dna[2])
                    @k_u

'RNAp-binding-bound' DNA(type{operator}, ds[1], bs[_]),
                     DNA(type{promoter}, us[1], bs[.]),
                     RNAp(dna)
                  -> DNA(type{operator}, ds[1], bs[_]),
                     DNA(type{promoter}, us[1], bs[2]),
                     RNAp(dna[2])
                  @k_b
```

Figure 6.11: Binding of RNA polymerase to a promoter with different rates, k_u and k_b according to context given by operator state.

6.4.2.1 Binding Rules with Context

The simplest kind of binding rule is just the same as unbinding with the direction of the arrow reversed. Such rules appear for the initiation of translation—the binding of a ribosome onto a ribosome binding site — as well as for the activation of an operator. These are not reproduced here. Instead, we consider binding rules with context, as in Figure 6.11.

The explicit context, with the operator adjacent to the promoter being bound to a protein, or not, allows for the modelling of inducible or repressible promoter architectures. The transcription process begins with the binding of RNA polymerase and the rate at which this happens depends on the state of the operators as illustrated in Figure 6.11. This is the simple case with only one operator but there is no restriction on the number




```
'coding-sequence-translation' RNA(type{X}, us[2], bs[.]),
                              RNA(ds[2], bs[1]),
                              Ribosome(rna[1], protein[.]),
                              .
                           -> RNA(type{X}, us[2], bs[1]),
                              RNA(ds[2], bs[.]),
                              Ribosome(rna[1], protein[3]),
                              P(type{X}, bs[3])
                           @k
```

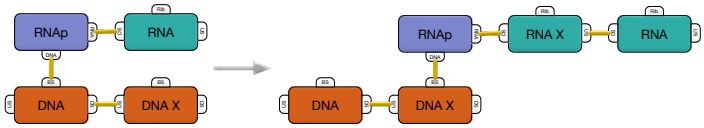
Figure 6.12: Translation of the RNA segment corresponding to a coding sequence to produce a protein.

of operators; we allow for upstream and downstream context of arbitrary size.

This example is illustrative in that rules are posed in terms of a "main" part that becomes bound or unbound and in principle it is possible to provide arbitrary amounts of context for *any* rule. This is supported by the low-level language here, but however it is only implemented in the compiler for the particular family of rules depicted in Figure 6.11, the activation of promoters through the binding of RNA polymerase. This is sufficient for models involving complex promoter architectures, but an extension allowing for context everywhere is not difficult.

6.4.2.2 Sliding Rules

The real work of modelling the transcription and translation machinery is done with sliding rules. Figure 6.12 shows how this works for the creation of a protein from a coding sequence. This is our first example of a rule where though the adjacent part figures explicitly in the rule, its *type* does not. It is sufficient to know that it is a piece of RNA. In this case, two pieces of RNA are involved, the part that is central to this rule corresponds to the coding sequence for X. It is adjacent to another piece of RNA, and the ribosome slides from one to the other (to the left, where sliding on DNA happens, as we will see next, to the right) and in the process, emits a protein of type X.




```
'transcription-elongation' DNA(ds[2], bs[1]),
                           DNA(type{X}, us[2], bs[.]),
                           RNAp(dna[1], rna[3]),
                           RNA(ds[3]),
                           .
                        -> DNA(ds[2], bs[.]),
                           DNA(type{X}, us[2], bs[1]),
                           RNAp(dna[1], rna[3]),
                           RNA(ds[4]),
                           RNA(type{X}, us[4], ds[3], bs[.])
                        @k
```

Figure 6.13: Transcription, production of an RNA sequence from DNA

A somewhat more complicated sliding rule than the one presented in the *Output Representation* section of the main text is used to implement transcription, as shown in Figure 6.13. This shares the feature of the translation rule above where there is a part that is central to this rule, part X, and there is an adjacent part whose type does not matter. Here, the RNA polymerase starts off bound to the adjacent DNA part, whose type does not matter and so is not specified, and slides onto the central part of type X. In the process, an RNA part of type X is inserted into the growing chain.

Other rules are necessary, of course. The rule in Figure 6.13, for example, cannot operate without a piece RNA bound to the polymerase. Chains of RNA cannot be produced before the first link has been added. The rule that does that is exactly analogous to that of Figure 8 in the main text. And similarly in the other direction, there is a rule to produce protein chains where a protein already exists and a coding sequence is slid across. This is almost identical to making an RNA chain. All of the other core rules are simply variations on those given above.

6.5 Genetic Circuit Compiler

Having described the GCDL in some detail, we now briefly sketch our implementation of the compiler. Many compiler implementations are possible; ours innovatively combines the logical inference that is native to the semantic web with the use of templates to generate the target program. The templates define standard models for each type of part in a given output language. Different output languages or model granularities are achieved by choosing a different set of templates. The overall information flow through the compiler is illustrated in Figure 6.1.

Our strategy is to first gather all the input statements and background facts that are asserted by the various vocabularies in use. In the first inference step, standard RDF rules are used to make available consequent facts that will be needed to produce the ultimate result. The result is a program in a language such as κ and not RDF, and which uses local variable names and not URIs, so the materialised facts are transformed into a suitable internal representation. Substitution into templates is done next, and finally the result is post-processed to derive any remaining program directives that are only knowable once the complete circuit is assembled.

It is interesting to consider that the entire compiler can be thought of as implementing a kind of inference quite different from what is commonly used with the Semantic Web. The consequent, the executable model, is in a different language from the antecedent, the declarative description. Through the use of embedding annotations, however, the original model is nevertheless carried through to the output, and is unambiguously recoverable. There is thus an arrow from the space of declarative models in RDF to the space of annotated executable models. There is an arrow in the other direction that forgets the executable part and retains the declarative part. In an important sense, the two representations contain the same information, only that the executable model has more materialised detail in order that it may be run.

6.5.1 Semantic Inference

The input from the user is the model description in the high-level language as described in Section 6.3. This description uses terms from, and makes reference to the `gcc` and `rbmo` vocabularies. The *meaning* of these terms, in the context of deriving an equivalent version of the program in the low-level language, is given by the companion inference rules. This is a somewhat subtle concept so let us illustrate what it means. Consider the statement,

```
:R0040a a gcc:Operator.
```

This statement gives the type of `:R0040a` as `gcc:Operator`.

The implications of this statement allows to identify the correct template to use for this part, found from information provided by the `gcc` vocabulary. Indeed, as a background fact, we have,

```
gcc:Operator gcc:kappaTemplate rbmt:operator.ka.
```

or in other words that an `gcc:Operator` corresponds to the template `rbmt:operator.ka`. We also have an inference rule, provided with the `gcc` vocabulary that says,

```
{ ?part a [ gcc:kappaTemplate ?template ] }
        => { ?part gcc:kappaTemplate ?template }.
```

In the Notation 3 (Berners-Lee; Connolly, et al., 2008; Berners-Lee; Connolly, 2011) language this means that, "for all `?part`s that has a type that corresponds to a kappa `?template`, that `?part` itself corresponds to that `?template`". Alternatively,

$$\texttt{type}(p,x) \wedge \texttt{kappa}(x,t) \rightarrow \texttt{kappa}(p,t)$$

It would have been perfectly possible to explicitly write what template should be used for each part in the high-level model description. That is not desirable because it would leak implementation details of the compiler into what ought to be an implementation-independent declarative description.

The above rule, and others like it serve to elaborate the high-level description into a more detailed version suitable for the next stage of the compiler and relieve the user of the need to supply the extra details. All implications that can be drawn under the `rdfs` inference rules and the `gcc` specific rules are drawn and become part of the in the in-memory RDF storage as the transitive closure of the rules (given the background facts and the provided model facts).

6.5.2 Internal Representation

The output of the first stage of the compiler contains all the information necessary to completely describe the output, but it is not in a convenient form for providing

to the template rendering engine. Our implementation choice for the compiler is the Jinja2 (Ronacher, 2008) rendering engine. This means that the appropriate data-structure is a dictionary or associative list that can be processed natively by these tools without need of external library. The required internal representation is built up by querying the in-memory RDF storage for the specific information required by the templates.

Our implementation does not require modification when new terms are added to the vocabulary and templates. To add support for a new kind of part it is necessary to write a new template for it and possibly add some terms to the vocabulary but does not require changing the compiler software itself. What makes this possible are the inference rules described in the previous Section 6.5.1. The queries on the RDF storage that produce the internal representation are posed in terms of the *consequents* of the inference rules rather than the specific form of input.

6.5.3 Template Substitution

The templates that produce the bulk of the low-level output are written in the well-known Jinja2 language. This language is commonly used for the server-side generation of web pages. KaSim or BNGL programs are not web pages but they are text documents and Jinja2 is well suited to generating them. It has a notion of inclusion and inheritance that is useful for handling the variations among the different kinds of parts, which typically differ in the rules for one or two of the interactions in which they participate with the others being identical. We provide a total of 15 templates for KaSim, of which there are top-level templates for each of the five distinct types of biological part defined in the `gcc` vocabulary as well as a generic part template, five templates implementing functionality shared among parts, and five consisting of supporting boilerplate required by KaSim.

A full description of the facilities provided by Jinja2 is beyond the scope of this paper, but a flavour is given in Figure 6.14 which shows an example of a template for a generic part (not having specific functionality like a promoter or operator might) demonstrating substitution of the `name` variable derived from annotation, and include statements referencing several other templates, one of which is reproduced and shows the KaSim code that is produced.

We use specific terms for defining the rates for the rules in which biological parts are involved, and a few other terms according to the function of the biological part of interest. It is possible to find the available terms out by inspecting the `gcc` vocabulary

(Appendix B.1).

A fragment of the `gcc` vocabulary is reproduced in Figure 6.15. Though this exposes some implementation detail, it is useful to understand the relationships between the various terms used to describe models. This is also important when supplying customised templates.

There are `gcc:Tokens`, so named because they correspond to tokens in the low-level language that are replaced. Each must have a preferred label that gives the literal token. In cases where there exists a sensible default value, this is given with `gcc:default`. The purpose of these statements is to act as a bridge between the fully materialised RDF representation of the model and the templates that require substitution of locally meaningful names.

For each kind of part (such as the `gcc:Operator` in the example in Figure 6.15), there are two main annotations that are necessary. For each machine-readable low-level language, a template is specified. The `gcc:tokens` annotations give the tokens that are pertinent to this kind of part. These must be specified in the high-level model or allowed to take on their default values. In addition to documenting the requirements of the templates for each kind of part, these statements are, "operationalised" and used by the compiler. They can equally well be used to check that a supplied high-level model is sufficiently complete and well-formed to produce an output program.

6.5.4 Derivation of Declarations

The KaSim language requires forward declaration of the type signatures of agents. This is by design (Jérôme Feret et al., 2015) so that the simulator can check that agents are correctly used where they appear in patterns in the rules. While this design choice can help a modeller that is writing a simulation program in the low-level language by hand, to assist in finding mistakes and typographical errors, it is not possible to know *a priori* what these declarations should be in the present context. The correct declarations for `DNA`, `RNA` and `Protein` depend on the complete set of parts that make up the model so their correct declarations cannot be in any template for an individual part.

To solve this issue, the compiler implements a post-processing step. The rules that are produced by instantiating the templates for each part are concatenated together with any explicitly supplied rules and then the whole is parsed. The use of each agent in each rule in this rule-set is assumed to be correct by construction. From there a declaration that covers each use of each agent is built up.

6.5.5 Initialisation

At this final stage of the compiler, all rules are present, both supplied by the user for the host environment and implied by the parts that form the genetic circuits and all declarations are also present. What is missing is the statement that creates an initial copy of the DNA sequence itself, which each upstream–downstream bond present. This information is, of course, available in the definition of the circuit, and so an appropriate `%init` statement, creating a single instance of the DNA sequence with correct linkages between the agent-parts is produced and added to the output. The low-level program is finally complete and ready to be executed.

6.6 Discussion

We have presented a language, the GCDL for describing genetic circuits and our compiler for generating simulation executables from it. We have made the case that the succinctness of the GCDL affords the user the benefit of describing the salient aspects of these circuits free of extraneous detail, that this reduces the potential for user error inherent in detailed coding of molecular interactions, and that this approach also affords flexibility in choosing the simulation or experimental methodology for the model. We have further developed the argument that modularity in modelling of genetic circuits has similar benefits of modularity in high-level programming languages, namely encapsulation and clarity. We now consider some of the limitations and benefits of our design choices and explore some areas ripe for future research.

It is important to understand the correctness and verification properties of the compiler and the GCDL. The GCDL is an RDF-based language and models are typically written in Turtle. The syntax (Cyganiak et al., 2014)(Prud'hommeaux; Carothers, 2014) on a concrete level is well defined and models that are badly formed will be rejected. The standard templates are documented in machine-readable form in the GCDL vocabulary. Annotations that are required for a given part type also cause the model to be rejected by the compiler if they are not present. But the compiler does not perform verification in terms of how the parts are composed. Users are free to choose any DNA parts and in any order. For example, a model that includes a coding sequence part without preceding promoter and ribosome binding site parts is allowed, though and the resulting model would emit no protein agents and perhaps not be very interesting. Verifying whether a given circuit expressed in the GCDL is accepted by a parts grammar (Cai; Hartnett,

et al., 2007; Pedersen et al., 2009) verification of part sequences is out of scope for the compiler but could be the subject of future work.

The expressive power afforded by the design choice of modularity—fixing the level of abstraction for a model—comes at a cost. Biological parts are considered as atomic units. While it is straightforward to model complex mechanisms like combinatorial logic operators and cooperative binding it is not straightforward to mix models in terms of the part abstraction with models of the underlying substrate. Phenomena that inherently involve the physical or chemical structure of the DNA molecule or the shape of a protein cannot be modelled directly and we are restricted to simply asserting that they occur or not at some rate. Similarly, while parts which share nucleotide sequences and may overlap can be marked as such with the `gcc:overlaps` term, this has no effect on the modelling. If the fact of parts overlapping is significant in the behaviour of the circuit, those parts are not modular and that would break the abstraction barrier. Such an annotation can, however, be used when selecting parts for assembly *in vitro*. Parts for which overlap is functionally significant can also be treated as an atomic unit with a suitable template. The modelling abstraction, once chosen, is fixed. This is by design, in order that models so expressed remain tractable.

Similar reasoning applies to optimisation of DNA sequences. This is not our focus in the present work. Here, our main goal is to capture the dynamics of genetic circuit designs and to automate the process of model generation. Hence, deriving final DNA sequences encoding the behaviours captured in models is not our focus, and related research can indeed be incorporated in the future (Mısırlı; J. S. Hallinan, et al., 2011). Because the language is based on RDF, custom user based data can be stored as annotations (Mısırlı; Cavaliere, et al., 2015) to facilitate later optimisation.

We do, however, envision optimisation of circuits at the level of abstraction that we have chosen, and derivation of circuits to a given specification. A method for doing this, which we only sketch here, is to define a suitable fitness or distance measure on the output of simulations with respect to the desired specification. A starting candidate circuit is chosen, constructed from a given library of parts, and measured. Parts of the circuit are swapped, added or removed at random, subject to the constraint that the circuit remains well formed according to an operon grammar (Cai; Hartnett, et al., 2007; Pedersen et al., 2009) and the new circuit is measured with respect to the specification. If the result is better than the previous circuit, the change is accepted, and the process is repeated until a locally optimal solution is found. This evolutionary algorithm approach is in contrast to the approach of assembling all possible circuits *in*

vitro seen elsewhere (Guet et al., 2002; Menzella et al., 2005; Smanski et al., 2014; Cress et al., 2015) and is likely to be less efficient in cases where the desired behaviour of the circuit can be measured simply, such as by detecting the production of a fluorescent protein. However for cases that may be more difficult to measure *in vitro* such as oscillations or more complex outputs it can be more straightforward to measure the output and compare to the specification when done *in silico*.

Currently, the templates that we have supplied only handle single stranded genetic constructs. Parts are composed using upstream and downstream bonds to create chains of DNA sequences, and our framework currently does not consider whether the other strand is free or not regarding the elongation RNAP or the binding of molecules and so on. One reason why we have chosen to support the single-stranded case first is simplicity. Another is that databases of models for double-stranded parts are not available. Adding support for this in templates, and developing a library of suitable parts is another topic for future research.

Here, we presented the application of rule-based models and Semantic Web technologies to automate the design of genetic circuits. Representations of cellular activities were captured using modular rules to support scalability of designs. The automation process is facilitated by the GCDL high level language, which is built upon the Semantic Web and is used to describe genetic circuits. Furthermore, we presented a compiler that generates rule-based executable models from the high-level description. The implementation of the compiler is notable in its use of semantic inference and the language is sophisticated enough to support several classes of complex regulatory mechanisms. Despite the expressive power afforded by this approach, the language maintains a succinctness and simplicity that we hope will be a boon to those modelling genetic circuits in silico. The implicit modularity in our rule-based approach and the high-level language presented will be beneficial to synthetic biologists to model complex regulatory relationships through the use of widely adopted standards.

```
## Auto-generated generic part {{ name }}
{% include "header.ka" %}
{% import "context.ka" as context with context %}
{% import "meta.ka" as meta with context %}

{% include "transcription_elongation.ka" %}
{% include "transcription_termination.ka" %}
{% include "translation_chain.ka" %}
{% include "translation_elongation.ka" %}
{% include "translation_termination.ka" %}
{% include "host_maintenance.ka" %}
```

```
{% set rule = "%s-translation-chain" % name %}
//
//^ :{{ rule }} a rbmo:Rule;
//^   bqbiol:isVersionOf go:GO:0006415;
{{ meta.rule() }}{# #}
//^   rdfs:label "{{ name }} formation of \
//^               translational chains, due to \
//^               gene fusion or leakiness of \
//^               stop codons".
// {{ name }} formation of translational chains,
// due to gene fusion or leakiness of stop codons
//
'{{ rule }}' \
    RNA(ds[2], bs[1]), \
    Ribosome(rna[1], protein[3]), \
    RNA(type{{ curly(name) }}, us[2], bs[.]), \
    Protein(ds[.], bs[3]), . \
-> \
    RNA(ds[2], bs[.]), \
    Ribosome(rna[1], protein[3]), \
    RNA(type~{{ name }}, us[2], bs[1]), \
    P(ds[4], bs[.]), \
    P(type{{ curly(name) }}, us[4], bs[3]) \
    @{{ translationElongationRate }}
```

Figure 6.14: Template examples. On top is the template for a generic part, and it references several other templates, one of which, `translation_chain.ka`, is reproduced on bottom.

```
gcc:transcriptionFactor a gcc:Token;
    skos:prefLabel "transcriptionFactor".
gcc:transcriptionFactorBindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorBindingRate";
    gcc:default 1.0.
gcc:transcriptionFactorUnbindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorUnbindingRate";
    gcc:default 1.0.

gcc:Operator rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:operator.ka;
    gcc:bnglTemplate rbmt:operator.bngl;
    gcc:tokens
        gcc:transcriptionFactor,
        gcc:transcriptionFactorBindingRate,
        gcc:transcriptionFactorUnbindingRate.

```

Figure 6.15: The specification in the gcc vocabulary of an gcc:Operator and associated terms.

Chapter 7

Guided Evolutionary Discovery of Genetic Regulatory Networks

Abstract. An important strategy for rendering the study of synthetic genomes tractable is modularity. Nucleotide sequences for operators, promoters, and coding sequences are considered as interchangeable atomic units with defined interactions. Given a library of such genetic parts, the design space of possible gene regulatory networks is combinatorially large. Building on our previous work on modular representation and rule-based simulation of genetic circuits, we propose an evolutionary algorithm guided by domain knowledge to explore this space *in silico* to obtain networks that satisfy a given specification and that can be interfaced, by means of appropriated meta-data, with automated biofoundries. We test the proposed computational framework on noisy oscillatory networks, an important class of synthetic circuits difficult to evaluate in vitro.

7.1 Introduction

Synthetic biology approaches genetic engineering by attempting to design biological systems, even entire genomes, from first principles (Mukherji et al., 2009; Baldwin, 2012). This approach has met with success in synthesis of drugs (Paddon et al., 2013; Galanie et al., 2015), production of biofuels (Ferry et al., 2012), treatment of disease (Ruder et al., 2011) and biological computing (Macia et al., 2014). The design

The work presented in Chapter 7 is joint work with Matteo Cavaliere and Vincent Danos. The work was conceived by all of the authors. I originated the concept of employing a grammar to guide the evolutionary search for synthetic genetic circuits that match a given specification, developed the mathematical formulation, implemented the `kgen` software performs the algorithm and drafted the text.

space of possible genetic circuits is combinatorially large (Somogyi et al., 1996). The transcription regulatory apparatus of genetic circuits is computationally powerful; it has been shown to be formally equivalent to the class of Boltzmann machines (Buchler et al., 2003). It is not known how to efficiently search this design space for solutions to particular problems, therefore circuits that have been studied are typically small and manually designed. We build on previous work using abstraction and modularity to constrain this design space and propose the use of an evolutionary algorithm to explore it to discover circuits with particular specified behaviours.

Abstraction and modularity, concepts imported from computer science where design components are encapsulated and considered as atomic units that interact only in well-defined ways, has been used to render the problem of synthetic genome design more tractable (Endy, 2005; Del Vecchio et al., 2008; Wang et al., 2011; Kittleson et al., 2012). There are efforts to standardise these modules and construct databases of descriptions (Canton et al., 2008; Li et al., 2010; Galdzicki; C. Rodriguez, et al., 2011) and models (Snoep et al., 2003; Moraru et al., 2008; T. Yu et al., 2011; Mısırlı; J. Hallinan, et al., 2014). Many of these *standard biological parts* are currently manufactured for use in a commercial or research settings.

When genetic regulatory networks are studied *in silico*, they are modelled and simulated numerically with deterministic or stochastic techniques (De Jong, 2002). Stochastic models are essential to capture the effects of noise which play an important role in natural and engineered biological systems (Balázsi et al., 2011). It is also infeasible in general to finitely represent molecular interactions in a chemical reaction formalism amenable to representation in differential equation form. To address these limitations, a more general *rule-based* formalism has been developed (Danos; Laneve, 2004; M L Blinov et al., 2008; Danos; Jérôme Feret, et al., 2008; Chylek; Leonard A Harris, et al., 2014) and software tools exist for stochastic simulation of models expressed in this way (Michael L Blinov; James R Faeder, et al., 2004; James R. Faeder et al., 2009; Lopez et al., 2013b; Boutillier et al., 2018).

Recent work has sought to bridge these simulation techniques with the model databases and automate the simulation process. An essential prerequisite is the ability to annotate models in an expressive, machine-readable and implementation-neutral way (Galdzicki; Clancy; Oberortner; Pocock; J. Quinn, et al., 2014; Madsen et al., 2016; Mısırlı; Cavaliere, et al., 2015; Cavaliere et al., 2019). In this way, knowledge about the biological parts is made available for use in automated assembly, both *in silico* and *in vitro*. The κ BioBrick Framework (KBBF) (Wilson-Kanamori et al., 2015) approach is

framed precisely to express stochastic models of biological parts in a generic way for simulation. We developed the Genetic Circuit Compiler (Waites; Mısırlı, et al., 2018) to bridge the description-simulation gap by generating simulation code from abstract annotated circuit descriptions.

Evolutionary algorithms for discovering regulatory networks and their parameters have been studied theoretically (François; Hakim, 2004; François; Siggia, 2008), including using rule-based modelling techniques (S. Feng et al., 2015). The latter approach (S. Feng et al., 2015) has the drawback that while it may discover a suitable circuit that can be simulated there is no guarantee that it can be constructed from real standard parts. Biological parts have been systematically assembled *in vitro* on plasmids (Guet et al., 2002), with the advantage that a large number of cells implement each combination in parallel. However the use of a simple threshold for selection (production of a fluorescent protein) is relatively insensitive and does not account for time variation (Ziv et al., 2007) or inspection of the intermediate products.

We describe an evolutionary algorithm for obtaining synthetic genetic circuits which can be constructed from a library of parts to meet a given specification. This algorithm is shown schematically in Fig. 7.1. The search for an optimal circuit is guided by the knowledge encoded in the semantically annotated descriptions of the parts in the library (Mısırlı; Cavaliere, et al., 2015; Waites; Mısırlı, et al., 2018) and by a grammar that generates a syntactically correct genome (Cai; Hartnett, et al., 2007; Pedersen et al., 2009). With a suitable grammar, evolutionary trajectories obtained using sequences of mutations that add or remove functionality to a circuit can be achieved. This combined approach, leveraging semantic annotation and syntactical constraints is key to obtaining results that can then be constructed *in vitro*. In particular, models appropriately annotated with meta-data can be interfaced with automated biofoundries. In this way, approaches such as the one we describe here become important tools for the design, testing, and construction methodologies in synthetic biology (Chao et al., 2017).

Our approach is novel in that it combines methods that have previously only been considered separately: guided evolutionary search together with annotated biological parts used to produce a stochastic simulation program. To our knowledge, guided evolution, which attempts to lower the total cost of identifying a circuit that performs well by constraining the search space and prioritising simpler circuits has not previously been applied to discovery of synthetic genomes. Neither has an annotated library of standard parts been used to further constrain the search space to only those circuits that are implementable *in vitro*. The use of rule-based stochastic simulation has the

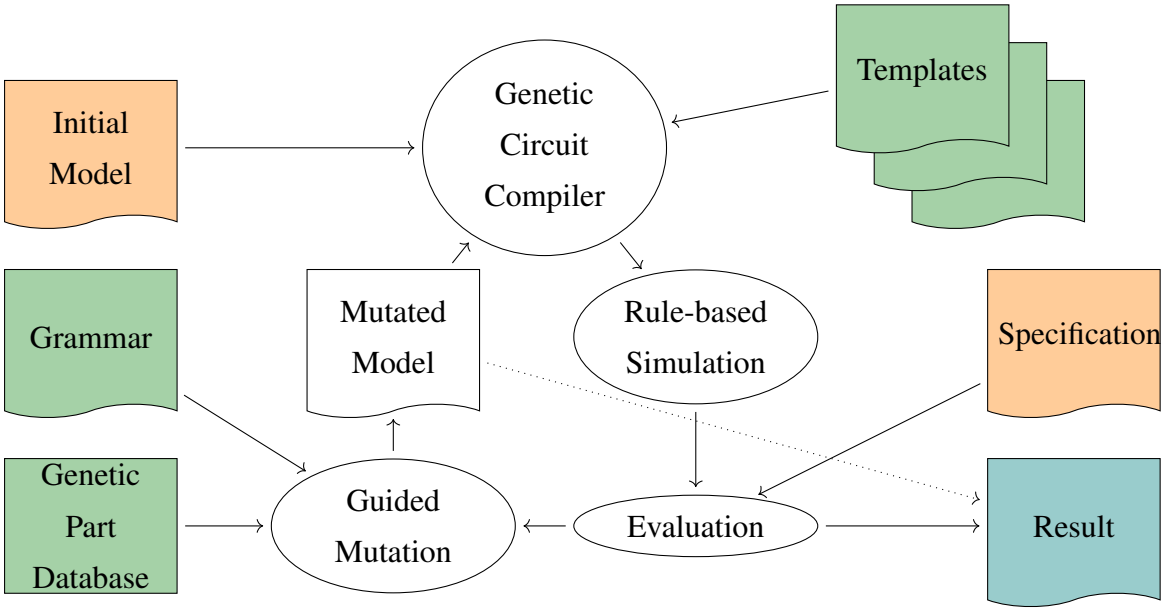


Figure 7.1: Workflow of the evolutionary algorithm. The inputs are shown in two groups. The initial model and the specification shown in orange are provided by the user. The genetic part database and implementation templates, as well as the grammar (which may alternatively be inferred from the initial model), shown in green, are typically fixed and used for different circuits. The evolutionary algorithm makes use of the Genetic Circuit Compiler, the KaSim rule-based simulator (Krivine et al., 2018), and implements mutation of the circuit and fitness evaluation on the output of the simulator to select the result, shown in blue. Results are annotated descriptions of synthetic genetic circuits which can be interfaced with automated biofoundries using meta-data such as SBOL (Madsen et al., 2016).

further benefit that it enables exploration of circuits under noisy conditions and with low protein copy numbers.

7.2 Background

The operation of our evolutionary algorithm builds heavily on previous work by ourselves and others on describing and simulating biological parts and genetic circuits. We provide a brief overview of the salient concepts and technologies here.

7.2.1 Regulatory Networks and Genetic Circuits

We briefly describe genetic circuits and regulatory networks, for those readers unfamiliar with the domain, otherwise it may safely be skipped. For more background on the subject, we recommend a text such as *Synthetic Biology: A Primer* (Baldwin, 2012).

The functional unit of a gene is the *operon*. A stylised example of the simplest kind is shown in Fig 7.2.

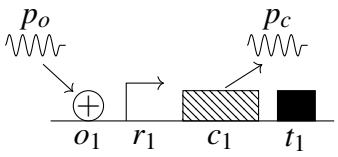


Figure 7.2: A stylised diagram of an operon.

This is composed of the sequence of parts, (o_1, r_1, c_1, t_1), representing an operator, a promoter, a coding sequence and a terminator respectively. This device functions as a gate. If protein p_o is bound to the operator, then the promoter initiates the transcription and translation machinery creates a new protein p_c of a kind determined by the coding sequence. The terminator causes the operation of the device to finish.

Promoters can be controlled by more than one operator enabling the construction of Boolean gates. The operator can be inductive, as in the above example, or repressive (which we would draw as $\ominus$) in which case the output protein is produced if it is *not* bound. More complicated operator-promoter architectures are equally possible. For example in co-operative binding with a pair of operators, one is much more likely to become bound only when the other is bound.

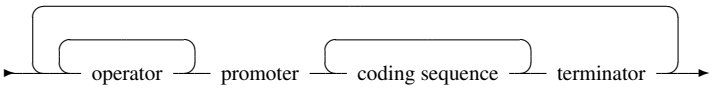
Boolean networks may thus be constructed from a set of operons, conventionally laid out in sequence, called a genetic circuit. The output protein produced by one serves as

input to the operator-promoter control gate of another. In addition, proteins may interact directly with one another, for example by binding to create new proteins. In principle, arbitrarily complex networks can be created. The operator-promoter conditional gate control and the fact that loops can be created with the output protein of a series of operons controlling the input means that, in principle, these networks have sufficient expressivity to compute a large class of functions. In practice, their size is constrained by the limited repertoire of proteins and operators.

Though it may provide a useful intuition for understanding genetic regulatory mechanisms, this account of genetic circuits is idealised in several ways. Each action of binding or unbinding proceeds in a probabilistic way, likely to happen or not at a given time according to the quantity of molecules available and the underlying kinetic rate. The picture of a promoter initiating transcription only if its operator is bound is more accurately described as initiating transcription at a much higher rate in that case. Proteins degrade at a certain rate. Importantly, input and output are encoded not as the presence or absence of a protein, but as the copy number of these molecules. Using threshold functions, a facsimile of digital behaviour can be recovered, but the underlying mechanism, in experiment and in simulation, has a far more analogue character.

7.2.2 Grammars for Genetic Circuits

The functional separation of biological parts implies that they should be assembled in a particular order. Grammars for this assembly are established (Cai; Hartnett, et al., 2007; Pedersen et al., 2009), and we use a simplified version here:



$\langle circuit \rangle ::= \langle operon \rangle \langle circuit \rangle \mid \langle operon \rangle$

$\langle operon \rangle ::= \langle opers \rangle \langle promoter \rangle \langle codes \rangle \langle terminator \rangle$

$\langle opers \rangle ::= \langle operator \rangle \langle opers \rangle \mid \langle operator \rangle$

$\langle promoter* \rangle ::= \langle promoter \rangle \langle ribosome\ binding\ site \rangle$

$\langle codes \rangle ::= \langle coding\ sequence \rangle \langle codes \rangle \mid \langle coding\ sequence \rangle$

According to this grammar, a genetic circuit is a sequence of one or more operons. An operon in turn consists of one or more operators, a promoter, a ribosome binding site, one or more coding sequences, and a terminator. The terminals for this language are the parts in the database. Their lexical class is identified by the type of the object.

7.2.3 Databases of Biological Parts

Biological parts from which a circuit may be built are represented in RDF (Cyganiak et al., 2014) in the database. The primary vocabularies used are the Rule Based Modelling Ontology (RBMO) (Mısırlı; Cavaliere, et al., 2015), containing terms for annotating rule-based models. The Genetic Circuit Compiler (GCC) vocabulary (Waites; Mısırlı, et al., 2018) designed for the specific purpose of representing implementation-independent models of genetic circuits from which simulation code may be derived. For readability, we write data of this kind in the Turtle language (Prud'hommeaux; Carothers, 2014).

An extract of the database is shown in Listing 7.1. It contains a selection of objects: proteins, and various biological parts. A complete description is found in Chapter 6, but we recap some features of note here.

Each object has a variety of rates associated with it. This information is primary motivator for having such a database. The rates of interaction between proteins, DNA, RNA, RNA polymerases and ribosomes are usually only known empirically and the functioning of regulatory networks depends crucially on them. The binding and unbinding rates are specified explicitly for the operator, and other rates take on a default value from the GCC vocabulary.

The most sophisticated object in the extract is the promoter. It has rates for RNA polymerase binding (which initiates the transcription process) according to whether the operator to its left is bound or not. It is possible to specify an arbitrary amount of such upstream or downstream context. This facility is available for operators as well in order to support cooperative binding and unbinding in more complex regulatory scenarios.

```
:P0040 a gcc:Protein;
  gcc:molecule "P0040";
  rdfs:label "TetR";
  gcc:proteinDegradationRate 0.0001.

:O0040 a gcc:Operator;
  rdfs:label "TetR activated operator";
  gcc:part "OTetR";
  gcc:transcriptionFactor :P0040;
  gcc:transcriptionFactorBindingRate 0.01;
```

```
    gcc:transcriptionFactorUnbindingRate 0.01.

:PrRM a gcc:Promoter;
  rdfs:label "Medium repressible promoter";
  gcc:part "PrRM";
  gcc:next "B0034";
  gcc:rnapBindingRate [
   gcc:upstream ([a rbmo:BoundState; rbmo:stateOf []]);
   gcc:value 7e-7
  ], [
   gcc:upstream ([a rbmo:UnboundState; rbmo:stateOf []]);
   gcc:value 0.0007
  ].

:C0040 a gcc:CodingSequence;
  rdfs:label "Coding sequence for TetR";
  gcc:part "CTetR";
  gcc:protein :P0040.
```

Listing 7.1: Extract from example parts database

Two crucial links can be seen on inspecting the operator and the coding sequence. For the operator, the protein that activates or deactivates it is indicated with the `gcc:transcriptionFactor` predicate. Likewise for the coding sequence, the protein that will be produced through transcription and translation is indicated with `gcc:protein`. These links, together with the rate information, are the foundation of the regulatory model.

7.2.4 Descriptions of Genetic Circuit Models

A starting point in the search for the target behaviour is provided by the user. This is done with an underspecified model (Listing 7.2). This model has the same form as a model of a complete circuit as described in Chapter 6, with some bibliographic metadata and other required scaffolding. What makes this model underspecified is the way that the circuit is written. Rather than the parts of the circuit being identified with specific entities using Uniform Resource Identifiers (URIs), they are indicated using *blank nodes* (Cyganiak et al., 2014).

```
:m a rbmo:Model; ## Model declaration
  dct:title "An two operon circuit";
  gcc:prefix <.../twooperon#>;
  gcc:include <.../host.ka>;
  gcc:init      ## initialisations
```

```
    [ rbmo:agent :RNAp; gcc:value 700 ],
    [ rbmo:agent :Ribosome; gcc:value 1000 ];
  gcc:linear (  ## The circuit is a list of parts
    [ a gcc:Operator ] [ a gcc:Promoter ]
    [ a gcc:RibosomeBindingSite ]
    [ a gcc:CodingSequence ] [ a gcc:Terminator ]
    [ a gcc:Operator ] [ a gcc:Promoter ]
    [ a gcc:RibosomeBindingSite ]
    [ a gcc:CodingSequence ] [ a gcc:Terminator ]
  ).
```

Listing 7.2: Two operon circuit example, with parts represented by blank nodes. Blank nodes are variables, indicated with square brackets, [...].

The form, `[ a gcc:Operator ]` in the Turtle language is equivalent to the existentially qualified statement, $\exists x.\textsc{Type}(x, \texttt{gcc:Operator})$. Each of these variables is thought of as a "blank" to be filled by parts from their database. The user thus provides an initial circuit skeleton with blanks to be filled in according to their type. The type information makes it possible to check the circuit against the grammar to ensure well-formedness. Such a description can be understood as a query against the database; the result identifies specific parts corresponding to the variables in the specification. A concrete circuit is obtained by randomly selecting from the result set for the query. Note that partial specification is possible: parts may also be specified as URIs, without the existential quantification.

7.3 Evolutionary Algorithm for Genetic Circuits

To discover circuits, we propose an evolutionary algorithm (Algorithm 1). Its operation is straightforward. It functions to fill in the blanks in the underspecified genetic circuit description. Each generation, a starting point or set of parent circuits are chosen (or supplied in the case of the first generation). The size of the population of circuits under test is fixed. The next generation is created with a combination of elitism and mutation. A fraction, γ of the next generation is obtained by selecting individuals from the previous generation proportionally to their fitness (see Section 7.4). The remaining fraction, $1-\gamma$, of the next generation is obtained by selecting individuals, again proportionally to their fitness, and mutating them. See Alg. 2 for a simple kind of mutation.

Using this mutation algorithm, the exploration of the space of possible circuits is guided by the syntactical structure of the starting circuit. Each mutation produces a circuit that has the same shape, the same sequence of types of parts. The permitted

Algorithm 1 Evolutionary Algorithm. This function performs one generation of the evolutionary algorithm. The function's argument, *parents*, is the set of individuals that make up the previous generation. There are three global parameters of the algorithm: *db*, the database of parts from which to assemble the circuit, β, the mutation rate and, γ the fraction of the previous generation to retain. Some circuits are retained from the parent generation using an elitist strategy, chosen proportionally to their fitness, SELECT(pop, γ). The remaining fraction of the next generation, $1-\gamma$, is created by selecting individuals from the population, again proportionally to their fitness, and randomly mutating them. Each offspring undergoes one or more mutations according to a sample from the exponential distribution with rate $\frac{1}{\beta}$. The function SAMPLE returns a random variable sampled according to the provided probability mass function.

```
1: function GENERATION(parents)
2:     children ← SELECT(parents, γ)
3:     for parent in SELECT(parents, 1 − γ) do
4:         child ← parent
5:         m ← CEIL(SAMPLE(1/β exp(x/β)))
6:         for 1...m do
7:             child ← MUTATE(child)
8:         APPEND(children, child)
9:     return children
```

Algorithm 2 Simple mutation. This function mutates the *parent* circuit by replacing one part with a randomly chosen one of the correct type from the global database *db*. It returns the mutated circuit.

```
1: function MUTATE(parent)
2:     i ← RANDINT(0, LEN(parent))
3:     kind ← TYPEOF(parent[i])
4:     child ← COPY(parent)
5:     child[i] ← RANDOMCHOICE(db, kind)
6:     return copy
```

evolutionary changes are restricted to swapping parts for others of the same type.

Because the mutation only changes one part at a time, it is possible, likely even, that the algorithm will become stuck in a set of circuits that are two or more changes away from a better performing circuit. Simply changing one part may not be sufficient to go from a non-functioning circuit to one that is even a little bit better: almost any fitness is unlikely to be a smooth function of circuits. To avoid getting trapped in these kinds of local minima, the evolutionary algorithm will occasionally mutate an offspring more than once, infrequently more than twice, and so forth. This is controlled by the mutation rate, β. The number of mutations is drawn from an exponential distribution with this rate. This strategy promotes exploration of a larger portion of the space of circuits while prioritising circuits similar to those that have been already tested and evaluated to perform well.

The question arises: is it possible for this evolutionary algorithm to produce an *infeasible* circuit, one that cannot be simulated? The answer is no, and this is true by construction. Every circuit is composed of modular parts that are linked together. The translation of parts to their corresponding rules is given by the templates supplied with the compiler. All parts in the database are translatable in this way. So every circuit produced by this algorithm can be represented as a set of low-level rules that can be interpreted by the κ-language simulator. There may, however, be circuits that cannot be represented in the rule-language in this way, either because they require behaviour that is not implemented in the templates or because the behaviour is somehow not rule-like. These circuits are not produced by the evolutionary algorithm at all because such parts are excluded from the database.

7.4 Test-Driven Evolution

To discover genetic circuits using the evolutionary algorithm, we adopt a practice analogous to that which is practice known as *test-driven development* (Janzen et al., 2005) in software engineering. Before writing the implementation of a given function, one first writes a test, or series of tests that checks that, for a certain input, the function should produce the desired output. If the tests are complete—rarely achievable in practice — the implementation of the function is considered correct. This process ideally begins with tests for the simplest function needed for the task at hand, and is repeated, gradually building up a library of functions needed to create the software program that is the ultimate goal. We take inspiration from this process for our task of

evolutionary genetic engineering.

In the setting of stochastic simulations of genetic circuits, we do not have the luxury of determinism. For all but the most trivial circuits, each simulation is likely to produce different results. This is the case even if the circuit under test is the best possible implementation of the desired behaviour. We can fix the input deterministically, but rather than specific outputs, we must consider the changing behaviour of the system in time. For stable circuits that enter a steady-state, the output specification can simply be an expected value of copy numbers of proteins of interest. For unstable or oscillating circuits, more sophisticated encoding is needed to specify the output.

We now give such a general framework suitable for evaluating the performance of synthetic genetic circuits *in silico*. Let C be the set of all possible genetic circuits and X be the set of multi-sets of agents. A simulation input, $x \in X$, is a multi-set representing the initial conditions, the copy number of each agent—transcription factor, RNA polymerase, ribosomes, etc.—that participates in the simulation. Let T be a set with a total order representing time. In the case of discrete simulations, it is usual to have $T \subset \mathbb{N}$, the natural numbers, or for continuous simulations $T \subset \mathbb{R}$, the reals.

A stochastic simulation is a non-deterministic function from circuits and agents to time-series of agents,

$$\mathrm{Sim} : C \times X \to (T \to X) \tag{7.1}$$

For a concrete instance of a simulation of a circuit, we write,

$$\mathrm{Sim}(c,x) = s_x(t) \tag{7.2}$$

To evaluate the performance of a circuit under test, we need initial conditions, x, a target object, ω, in some measured space Ω and a function, $\gamma : (T \to X) \to \Omega$ that takes a time-series and maps it into the same measurable space. Let $d_\omega : \Omega \to \mathbb{R}$ be the function that computes the distance in Ω to the target object. We can then define the general fitness function for that particular target as the expected value,

$$\hat{f}(c,x,\omega) = -E\left[d_\omega \circ \gamma \circ s_x\right] \tag{7.3}$$

This is just the average distance from the image through γ of simulated system state s_x to the target object ω. The use of a measured space is so that this distance is well defined. The sign is chosen to be negative to follow the semantics that "fitness" is a quantity to be maximised.

Calculating the expectation of Eq. 7.3 is straightforward if the system is ergodic. If there is no absorbing state, there is a non-zero chance of reaching any state from any

other eventually. Von Neumann's mean ergodic theorem (Neumann, 1932) tells us that in this case that the time average of a single trajectory and the expectation of the system as a whole are the same. As a consequence we have,

$$\hat{f}(c,x,\omega) = -\lim_{T\to\infty} \frac{1}{T} \int_0^T (d_\omega \circ \gamma \circ s_x)(t)\, dt \tag{7.4}$$

To account for the possibility of modelling non-ergodic systems or choices of Ω or γ that cannot be expressed in terms of a time integral or sum, Eq. 7.3 must simply be computed directly by averaging over an ensemble of many simulations at some point in time. A steady state is one in which the chance of the system leaving a given region of state space is arbitrarily small. This condition can be observed. The expectation is then calculated In this way one can support population-level evaluations of genetic circuits rather than simply evaluating individual instances for both ergodic and non-ergodic systems.

Test cases are pairs supplied by the user, $(x,\omega) \in U$ with $U \subseteq X \times \Omega$. This allows us to define the fitness of a circuit as the sum of Eq. 7.3 over all test cases,

$$f(c) = \sum_{(x,\omega)\in U} \hat{f}(c,x,\omega) \tag{7.5}$$

This quantity is used in Alg. 1 (line 6) to score circuits and select the most promising candidate for further evolution.

7.5 Examples

7.5.1 Bi-stable Switches

One of the simplest circuits that can be generated is the bi-stable switch (Gardner et al., 2000). A bi-stable switch is realised as a pair of operons arranged in a mutual negative feedback, shown diagramatically in Fig. 7.3.

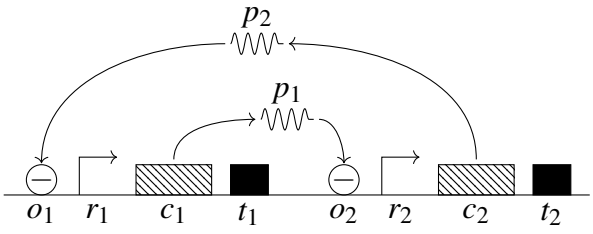


Figure 7.3: A bi-stable switch.

When the protein p_1 is being produced, it inhibits the production of p_2 and vice-versa. Starting from a state where neither protein is present, both will be produced at first but soon the system will settle into a state where either one or the other is produced exclusively. The chance of this being p_1 or p_2 depends on the rate that these proteins bind to o_1 and o_2 and the rate of RNA polymerase binding to r_1 and r_2, assuming that all other rates are equal in both operons. If the circuit is in the state where it produces p_1 and a quantity of p_2 is transiently introduced, the production of p_1 is suppressed and it will, with high probability, switch to the state where it is producing p_2. Thus the circuit has two stable modes of operation and behaves like a switch whose state is toggled by transient introduction of one or the other protein.

This kind of circuit is well known and studied. We do not need a genetic algorithm to find it. We will use it as a test case, where we know the answer in advance, to show how our input/output encoding works. Our challenge is to produce a specification that will guide the evolutionary algorithm to discover it. We assume, of course, that it is possible to construct this circuit from the parts available. We give a solution for this simple case.

Inspection of the above circuit suggests that when in some state, $s_i, i \in 1,2$, the proteins will be produced at some rate $\bar{k}_i - k_{d_i}$, where $\bar{k}_i$ is the overall production rate of the operon and k_{d_i} is the degradation rate for the protein. So long as $\bar{k}_i > k_{d_i}$, this means that the copy number of the protein will continue to increase with time, and there is no single number or distribution of the proteins themselves that can be used independently of time to characterise the circuit. We therefore define a functions of the copy numbers,

$$\begin{aligned} q_1(p_1,p_2) &= \frac{p_1}{p_1+p_2} \\ q_2(p_1,p_2) &= \frac{p_2}{p_1+p_2} \end{aligned} \tag{7.6}$$

and we expect that at steady state the following should hold,

$$\begin{aligned} i_1 = [n,0] \quad &\rightarrow \quad o_1 = [1,0] \\ i_2 = [0,n] \quad &\rightarrow \quad o_2 = [0,1] \end{aligned} \tag{7.7}$$

where input, i_i, are the initial copy numbers of p_j ($n > 0$) and o_i are the desired outputs for each test case i.

Now, given a circuit to test, we can fix the input and measure the extent to which its output differs from the specification. If e_i are the empirical results obtained by calculating q_j for simulation at steady state, they can be compared to the specification

with some appropriate measure, for example the sum of their Euclidean distances,

$$f(\mathbf{e}) = \sum_i \|e_i - o_i\| \tag{7.8}$$

and used as a fitness function.

7.5.2 Noisy Oscillators

In practice, bi-stable switches of the kind described above are not stable in the long-term because the transcription and translation machinery is noisy. An operon in the suppressed state is not simply inert; it expresses its protein at a very low rate. This means that it can transiently express a protein that will then flip the switch into the opposite state. The long-term behaviour of the circuit, absent any external influence, can both be independent of the initial state and bimodal. This phenomenon is clearly illustrated in Fig. 7.4a, which shows just such an unstable switch. It oscillates irregularly between its states of expression of one or the other proteins (*LcI* and *LacI*).

To capture this behaviour, we can reason that as the time-series are anti-correlated, a suitable fitness function can be constructed from the normalised cross-correlation,

$$f(\mathbf{p}) = -(\overline{p_{\mathrm{LcI}} \star p_{\mathrm{LacI}}})(0) \tag{7.9}$$

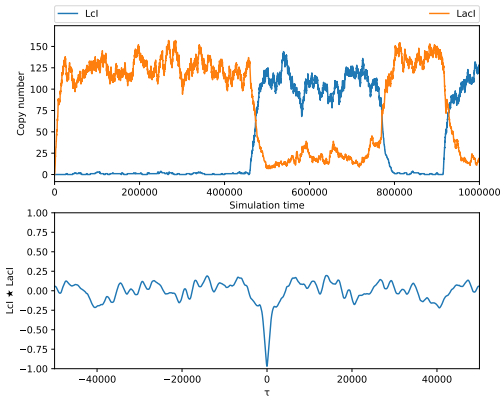
where, for two time series $a(t)$ and $b(t)$ with means μ_a, μ_b and standard deviations σ_a, σ_b respectively. The standard discrete cross-correlation is normalised as follows,

$$(\overline{a \star b})(\tau) = \frac{1}{n\sigma_a\sigma_b} \sum_n (a(n+\tau) - \mu_a)(b(n) - \mu_b) \tag{7.10}$$

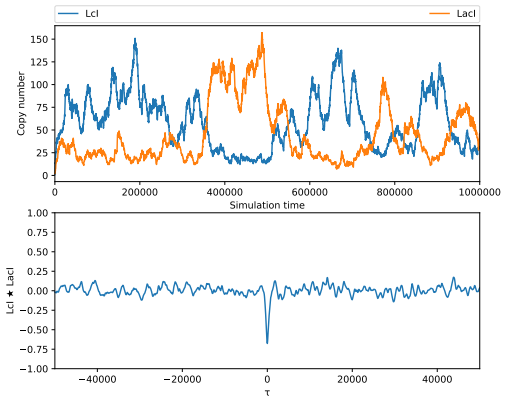
Note that the provided fitness function is expressed in terms of statistical properties of the simulation observables over time. This illustrates an important benefit of our approach. Not only can circuits be evaluated based on the quantity of products that are not directly observable in the laboratory, but their time-varying behaviour can be incorporated as well.

We used an example database containing coding sequences for the *LacI*, *TetR* and λ-*cI* proteins as well as the corresponding operators, inductive and repressive promoters acting at different rates, and generic ribosome binding sites and terminators. Using the fitness function given by Eq. 7.9 with the evolutionary algorithm Alg. 1, we discovered two families of circuits.

The first family corresponds to the mutually repressive circuit described above. An exemplar of this kind of two-operon circuit whose behaviour is shown in Figure 7.4b is,



(a) *LcI* induces *LcI*, *LcI* represses *LacI*



(b) *LcI* represses *LacI*, *LacI* represses *LcI*

Figure 7.4: Simulation trace of two varieties of unstable switch or oscillator. Notice the characteristic anti-correlation of the two proteins, *LcI* and *LacI*. Figure 7.4a shows an inducer-repressor circuit with sharp transitions between stable states and exhibiting nearly maximal anti-correlation, $(p_{\mathrm{LcI}} \star p_{\mathrm{LacI}})(0) \approx -1$. Figure 7.4b shows a mutually repressive circuit exhibiting strong anti-correlation, $(p_{\mathrm{LcI}} \star p_{\mathrm{LacI}})(0) \approx -0.7$.

```
OLcIa PrRFb B0034c CLacId B0011e
OLacIf PrRFg B0034h CLcIi B0011j
```

Sequence represents parts from the database. Those beginning with `O` are operators, `P` are promoters, and `C` are coding sequences. Those beginning with `B` are parts that are required for the circuit to function, ribosome binding sites and terminators, but do not interact with proteins and so do not govern the interaction between operons. The lower-case suffix in the part labels represents the position in the circuit and enables distinguishing between parts of the same kind that occur in multiple positions. Circuits of this kind, *A represses B, B represses A*, exhibit a fitness in the range of 0.4-0.7 as measured with Equation 7.9. The difference in fitness between circuits in this range is explained by the rates of the chosen promoters: faster promoters produce a stronger anti-correlation.

The second family is of inducer-repressor type. The exemplar shown in Figure 7.4a is,

```
OLcIa PrIFb B0034c CLcId B0011e
OLcIf PrRFg B0034h CLacIi B0011j
```

Circuits of this kind, *A induces A, A represses B*, exhibit a very high fitness typically between 0.8-1.0, likewise as measured with Equation 7.9. The operation of this circuit is

less obvious. Initially, there is no *LcI* present, and so *LacI* is produced rapidly. A small amount of *LcI* is produced which halts production of *LacI* and feeds more production of *LcI*. Eventually, by chance, no *LcI* is produced for a time and state of the system reverts to the initial state. Unlike the mutually repressive family of circuits where quasi-periodic behaviour might be expected, the oscillatory behaviour of the inducer-repressor family can mainly be attributed to noise. Again, the magnitude of the fitness, the strength of the anti-correlation is a result of the rates of the promoters.

The existence of multiple solutions is a feature: a primary task of the user of an evolutionary algorithm such as this is to design an appropriate fitness function according to their needs. If the fitness function is constructed so as to admit a larger set of acceptable results, this can permit the discovery of several circuits with equivalent behaviours *in silico*. Recall that the models of biological parts used for simulation are idealisations; having several alternatives to try in the laboratory is useful.

7.6 Software Implementation

The software with which the above results were produced is distributed with the `krdf` Python package[1] which also contains the implementation of the Genetic Circuit Compiler. The infrastructure for executing our evolutionary algorithm consists of several pieces:

ksim A worker process that receives model descriptions, simulates them using the KaSim software, and returns the resulting time-series.

kq A load-balancing server that receives requests to simulate circuits and distributes them among **ksim** workers.

krun A simple client program that requests excution via **kq** of a single, fully specified model.

kgen The implementation of the evolutionary algorithm as described above, that requests simulations via **kq**.

This distributed architecture enables simulation of circuits for an entire population to proceed in parallel.

[1] Available at: https://github.com/rulebased/composition

The results reported above were obtained with the example parts database (`partsdb.ttl`) and two-operon partially specified circuit (`twoop.ttl`) provided in the `examples/` sub-directory.

7.7 Performance Evaluation

As a baseline, we compare the performance of our evolutionary algorithm to a random search of the space of possible circuits to find a suitable circuit. By "suitable", we mean an inducer-repressor or a mutually repressive circuit that exhibits a fitness in the range 0.65-0.8. We searched the space of possible circuits by beginning with a population and randomly mutating circuits at each generation with no regard to fitness at all. We performed these random searches for population sizes of between 5 and 50 individuals with a variety of mutation rates from 0.25 to 1.5. For each combination of population size and mutation rate, we repeated the random search 100 times. We found that on average 483 distinct circuits must be evaluated, with a standard deviation of 40 globally and 25 within a given population size. Though population size and mutation rate influenced the number of generations required to find a suitable circuit, these parameters had no appreciable effect on the number of distinct circuits that needed to be evaluated.

We then performed a similar experiment using our evolutionary algorithm, executing the algorithm for a maximum of 1000 generations, for a variety of population sizes, fraction of the population retained in an elitist manner between generations, and mutation rates. For each set of parameters, the experiment was repeated 250 times. The main results are reported in Fig. 7.5.

We first consider, in Fig. 7.5a, the number of distinct evaluations required to find a suitable circuit. The effect of elitism is shown in the figure, where the horizontal axis shows γ, the fraction of the previous generation retained in the next. The trend is clearly that fewer evaluations are required if a greater fraction of the population is retained. The reason for this is straightforward: retained elite individuals are not simulated again, so the more that are retained, the fewer circuits must be simulated. As we will see below, this result is not quite as encouraging as it might at first appear.

The number of generations, shown in Fig. 7.5b, increases with the fraction, γ. As more individuals are retained in the population, there is less change with each generation. Therefore it takes more generations to find a suitable circuit. This shows how γ can be understood as analogous to a population growth rate in settings where the population

size is not fixed. A high growth rate corresponds to a a low γ as a relatively greater proportion of each generation consists of new individuals.

The dependence of the success rate on γ, shown in Fig 7.5c, is revealing. The more of the previous generation that is retained, the lower the success rate of finding a suitable circuit. This observation tempers the result from the previous Fig. 7.5a. With a low population turnover, i.e. a larger value for γ, there is a higher chance of getting stuck and producing no result at all. Greater elitism yields a shorter path to success, but with a higher chance of failure. Fig. 7.5d shows the result of correcting the number of evaluations by dividing by the success rate. The result is somewhat noisy, but it shows that the population size does not strongly influence the number of circuit evaluations required.

Overall, we can see it typically takes 150 evaluations of distinct genetic circuits in order to find a suitable one. This represents a speed increase of 60-70% over a random search. The number of generations required to find a solution depends quite strongly on the population size, the degree of elitism and the mutation rate. However because the simulation of genetic circuits is vastly more computationally intensive than the evolutionary search itself. A single simulation of a two-operon circuit can take from 30 seconds to several minutes, and simulations are conducted multiple times to obtain an average fitness. When a previously simulated circuit is encountered as a result of mutation, the previous result is simply used rather than simulating it again. The running time of the algorithm depends therefore far more strongly on the proportion of the space that is explored than on the number of generations. This is the reason why the running time is not especially sensitive to the parameters of the evolutionary algorithm.

We also observed during the course of these experiments that, in addition to finding the best circuits our simulations also reliably produced less effective yet functional circuits differing mainly in the speed of the promoters in the course of the search. This characteristic of exploring the neighbourhood of functional circuits is useful when we remember that our simulation technique is based on an idealisation and real circuits assembled *in vitro* may well display different behaviour. Providing a short list of candidate circuits for laboratory experiment is more important than finding an exact local optimum in the fitness landscape.

7.8 Extension: Syntactical Mutation

While the simple mutation above is sufficient to search for regulatory networks where the structure of the underlying circuit is given in advance such as (Guet et al., 2002), it is possible to generalise it and enlarge the space of circuits that can be explored. We sketch, but do not implement, how such a generalisation of our evolutionary algorithm could work.

Suppose now that each mutation operation can, with some probability, change the *type* of a part before selecting which part will be selected to produce a new candidate circuit. We extend the grammar given above in a way that supports the incremental addition of functionality but which does not, as a matter of syntax, destroy functionality. Specifically, we introduce a new kind of part, an inert spacer, and allow padding between operons. The addition of such spacers at the beginning of an operon enables its extension to the left with additional operators. We also allow terminators on their own to constitute an inert operon. This facilitates a sequence of mutations that results in an operon with additional coding sequences for cases where fused proteins are required,

$$\begin{aligned}\texttt{OP},\texttt{PR},\texttt{RBS},\texttt{CS},\texttt{T},\texttt{SP} &\rightarrow \texttt{OP},\texttt{PR},\texttt{RBS},\texttt{CS},\texttt{T},\texttt{T}\\ &\rightarrow \texttt{OP},\texttt{PR},\texttt{RBS},\texttt{CS},\texttt{CS},\texttt{T}\end{aligned}$$

where the abbreviations for part types have the obvious meaning (`OP` for operator, `PR` for promoter, and so forth).

The new and modified production rules required to accomplish this are shown below. With this modified grammar, an upper limit on the size of the circuit measured in number of parts is established by the user-provided underspecified model, and the number of operons is fixed, but they may grow to fill the available space to the left (operons) and to the right (coding sequences).

$$\begin{aligned}\langle \textit{circuit}\rangle ::= \ & \langle \textit{spaces}\rangle\ \langle \textit{operon}\rangle\ \langle \textit{circuit}\rangle\\ & \mid \langle \textit{spaces}\rangle\ \langle \textit{operon}\rangle\ \langle \textit{spaces}\rangle\end{aligned}$$

$$\begin{aligned}\langle \textit{operon}\rangle ::= \ & \langle \textit{opers}\rangle\ \langle \textit{promoter*}\rangle\ \langle \textit{codes}\rangle\ \langle \textit{terminator}\rangle\\ & \mid \langle \textit{terminator}\rangle\end{aligned}$$

$$\langle \textit{spaces}\rangle ::= \langle \textit{spacer}\rangle\ \langle \textit{spaces}\rangle \mid \langle \textit{empty}\rangle$$

The corresponding mutation algorithm is shown as Alg. 3. The type of a part is

Algorithm 3 Syntactical mutation. This function mutates the syntax of the input *parent* circuit with some probability p by randomly selecting a new type (kind) of part from the database *db*. It then replaces a part in the circuit with a randomly chosen part of the new type.

```
1:  function SyntaxMutate(db, parent)
2:      i ← RandInt(0, Len(parent))
3:
4:      if Rand() < p then
5:          kind ← RandomChoice(db, type)
6:      else
7:          kind ← TypeOf(parent[i])
8:      child ← Copy(parent)
9:      child[i] ← RandomChoice(db, kind)
10:     return child
```

mutated with some probability before the part itself is mutated. As this may produce a circuit that is not grammatical, this must be checked before it is returned as a candidate to the evolutionary algorithm,

Algorithm 4 Syntactical mutation. This function mutates the input *parent* circuit using information from the database, *db*. It does this by calling SyntaxMutate above and checking that the result is accepted by the circuit grammar. If the resulting circuit is accepted, it is returned.

```
1:  function Mutate(db, parent)
2:      repeat
3:          child ← SyntaxMutate(db, parent)
4:      until Accepts(grammar, child)
5:      return child
```

A further generalisation of the grammar to allow incremental construction of operons is equally possible. One approach is to first allow an operator and a terminator with any number of intervening spacers. From there, the circuit is built up from the left, first adding a promoter, then ribosome binding site and coding sequences, sequences of

transformations such as the following,

$$\begin{aligned} \texttt{SP,SP,SP,SP,SP} &\rightarrow \texttt{OP,SP,SP,SP,T} \\ &\rightarrow \texttt{OP,PR,SP,SP,T} \\ &\rightarrow \texttt{OP,PR,RBS,SP,T} \\ &\rightarrow \texttt{OP,PR,RBS,CS,T} \end{aligned}$$

It is straightforward to extend the grammar appropriately to include this evolutionary trajectory.

7.9 Discussion

In this chapter we have described several novel contributions to the design of genetic circuits. The evolutionary algorithm that we have outlined is guided by a starting design encoded in predicate logic using data in semantic web form. The design is mutated according to a grammar chosen to not only accept only valid genetic circuits but to support an evolutionary pathway to progressively more complex circuits from a simple starting point. The fitness function at the core of the evolutionary algorithm operates not on single measurements but on time-series measurements of proteins and other agents, enabling high fidelity comparison of *in silico* numerical results to the provided specification. By working with time-series produced by simulations we are able to account for noise, time-varying behaviour and population-level statistics which are important issues in the engineering of biological systems (Andrianantoandro et al., 2006).

Our evolutionary algorithm for discovering synthetic genetic circuits begins with an underspecified model description and a database of parts that can be used to complete the model. This approach showcases the strength of using an RDF representation for both. The use of blank nodes, understood as existentially quantified variables in the description, representing parts that are to be substituted, is intrinsically compatible with the content of the database. Indeed the underspecified model can be thought of as a query against the database which is satisfied in a specialised way using the evolutionary algorithm. Given an accurate database of this kind, because all of the parts in it are annotated with metadata that refers to the real-world entities to which they correspond, in principle there would be enough information to construct and validate the results of this query *in vitro*. Such an accurate database does not, however, exist today, and validation by laboratory experiment remains as an important line of future research.

Noise is an inherent quality of genetic circuits and indeed any molecular interaction where copy numbers are small. A limitation of laboratory experiments is that it is difficult to accurately measure these time-varying dynamics directly (Purnick et al., 2009). The strength of stochastic simulations of molecular interactions is that the entire system can be inspected and traced and distributions of any agent can be studied without the need for indicator molecules. Noise can be accurately characterised and measured and taken into account when comparing the behaviour of genetic circuits. The development of this capability in the context of the pipeline for discovering genetic circuits that we have described is novel.

The abstraction of biological parts as atomic entities that interact only in well-specified ways is a useful strategy to obtain tractable models for simulation. However is important to recognise the limitations of the approach. In particular, no attempt is made to model cross-talk or erroneous binding which, in real systems, can radically change the behaviour. It is assumed, for example, that either a given operator is bound by the *LacI* protein or it is not. No provision is made for the possibility that some other protein could bind to it with a different affinity and thus induce (or suppress) transcription at the adjacent promoter. Because the simulation environment is idealised, results must then be validated in the laboratory using real biological parts. It is for this reason that the procedure described here is designed to produce a set of candidate circuits rather than a single solution. In particular once a plausible circuit has been located, it is valuable to explore the genetic neighbourhood of that circuit because a related circuit that performs well but sub-optimally in simulation may indeed perform ideally in the laboratory.

This efficiency of the evolutionary algorithm compared to a random search depends crucially on the fitness function. The fitness landscape can be very discontinuous with respect to some kinds of changes and smoother with respect to others. A single change to a coding sequence in a mutually repressive circuit, for example, produces a radical change in overall fitness. On the other hand, replacing a promoter by a faster or a slower variant does not fundamentally alter the functioning of the circuit but will alter the response to the presence of whichever transcription factors govern its activation. The conclusion is that once basic regulatory structure has been established, there is value in optimisation by exploring closely related circuits, but exploring near relations of non-viable circuits has limited value.

There are additional opportunities for guiding the search for circuits that match the given specification. Above, we demonstrate searching a space of circuits constrained by a grammar, and additionally by partial specification if desired, where some parts

are varied and some held fixed, and the performance of the circuit is then scored. Absent syntactical mutation, mutated circuits are drawn uniformly from the set of circuits that differ in exactly one place. The random selection need not be uniform. For example, if a circuit already contains an operator-promoter segment activated by some protein, a change that would produce a second operator-promoter segment activated by that same protein may be penalised to avoid duplication. Mutated circuits could be selected proportionally to some monotonic function of their symbol entropy, for example, to prioritise searching of more heterogeneous candidates and avoiding those with redundant parts.

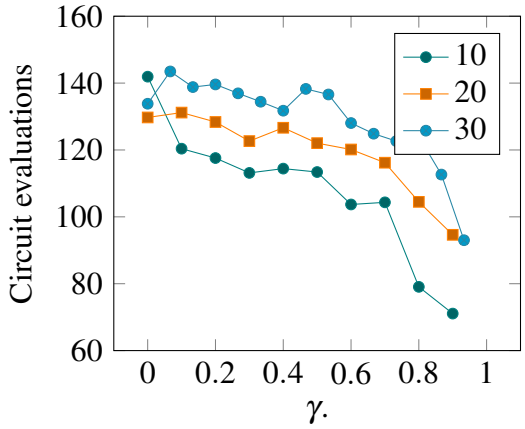
A note is in order about the generalisation mentioned at the end of Section 7.8: evolutionary trajectories adding or removing functionality to circuits. The general principle that simpler solutions should be preferred over more complex ones for the same problem should be reflected in the evaluation of circuits. A simple choice is to assign a circuit cost proportional to the number of (non-spacer) parts in the circuit. Another choice that accounts for the types of parts used is to use a function of the Shannon entropy (Shannon, 2001) of the circuit. The precise details of how best to combine such a measure of circuit complexity with the measures that we give above for circuit performance, and the behaviour of circuits whose genomes are permitted to evolve in this way, is an interesting area for future research.

It should also be emphasised that though we have only considered cross-correlation in detail, the framework that we have described admits more general functions of the simulation state. One interesting possibility is to consider probability distributions of the values in the simulation time-series compared with a reference distribution. There exist measures useful for comparing probability distributions, among them the Wasserstein distance (Givens et al., 1984) and the Kullback-Leibler divergence (Kullback et al., 1951). Working with directly with probability distributions over system states rather than summary statistics suggests that we may provide the evolutionary algorithm with more nuanced descriptions of the desired behaviour of genetic circuits and how their products might interact. This is another topic suitable for future research.

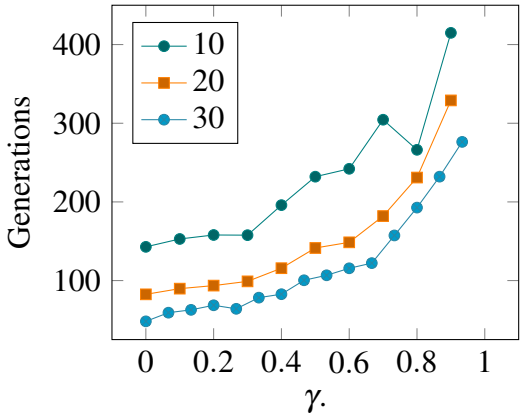
In this, and previous papers, we have been concerned mainly with engineering of regulatory networks at the nucleic acid level, describing synthetic genetic circuits where the principle underlying the computation is production of transcription factors which then induce or inhibit the production of other transcription factors. In our circuit compiler work (Waites; Mısırlı, et al., 2018), the question of protein-protein interactions was left to the side, to be provided by the user. However, recent work has shown

that it is equally possible to compute with proteins (Gao et al., 2018). It should be relatively straightforward to express that class of model within the present framework. Protein fusion, resulting from adjacent coding sequences for degrons, cleavage sites and the proteins themselves are readily represented at the genetic circuit level, and the interactions of the resulting fused proteins are systematic so amenable to templating for simulation. The entire system, now supporting programs in the genetic regulatory network language and the protein-protein interaction language, would be brought into scope for evolutionary algorithms such as the one presented in this paper. In addition to these two computational environments, there are others, among them Non-Ribosomal Peptide Synthases (NRPS) (Cane et al., 1998; Fischbach et al., 2006; Strieker et al., 2010), CRISPR/Cas systems (Cong et al., 2013) that modify the genetic material itself, synthetic guanine exchange factors (Yeh et al., 2007) and possibly others.

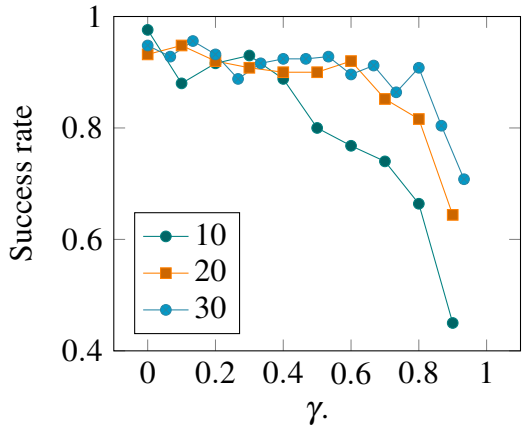
Our paper suggests a possible use of rule-based models to facilitate the automatic discovery of synthetic circuits annotated with meta-data that facilitate their *in vitro* implementation. However, this leads to the general question concerning the trade-off between the different formal languages for synthetic biology and the most appropriate ways to interface them with the emerging area of synthetic biology automation (Chao et al., 2017).



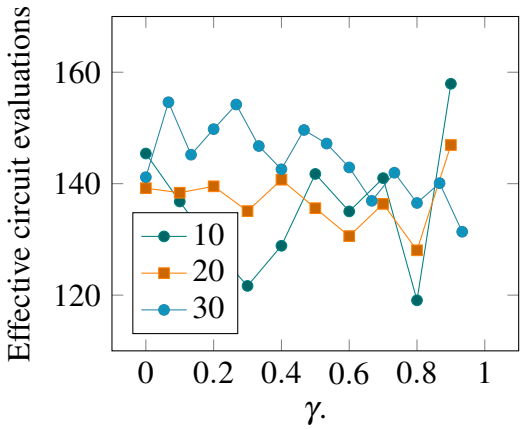
(a) Average number of distinct circuits evaluated, as a function of the fraction of elite individuals retained, γ, for various population sizes, n.



(b) Average number of generations, as a function of the fraction of elite individuals retained, γ, for various population sizes, n.



(c) Simulation success rate, as a function of the fraction of elite individuals retained, γ, for various population sizes, n.



(d) Average effective number of distinct circuits evaluated, corrected for the failure rate, as a function of the fraction of elite individuals retained, γ, for various population sizes, n.

Figure 7.5: Results of evaluating the evolutionary algorithm. We consider the following required to find an inducer-repressor or mutually repressive circuit: the number of evaluations of distinct circuits, the number of generations, the success rate of the algorithm and the effective number of evaluations taking into the success rate.

Chapter 8

Conclusion

If there is one theme that is consistent throughout this thesis, it is the role of measurement as applied to modelling. Chapter 2 describes a general framework for representing models, which provides a better understanding of the subject matter that is being measured. Chapter 3 develops the concept of *Path Entropy* to measure the agreement between a model of epithelial tissues and experimental results. Next, Chapter 4, generalises this measure to segmented 3D empirical models, developing the concept of *Structural Entropy* to quantify the degree of anatomical organisation in developing organisms. Chapters 5 and 6 contain an extended excursion, developing an infrastructure for modelling genetic circuits. Chapter 7 returns again to measurement, using the infrastructure developed in the previous chapters to simulate genetic circuits and measure their performance relative to a specification to find the best performing circuit. Mathematical and computational models are useful to the extent that they correspond to the phenomena they purport to represent. Defining measures to determine that extent in a principled way has been a central activity of this thesis.

There are several distinct contributions in this thesis. The first, a language of *propagators*, sets the stage by describing computational simulations in general. Though the original motivation was as an *aide-mémoire*, to provide a succinct yet accurate way to communicate the behaviour of simulation programs written in languages such as C or C++, it can also serve as a specification against which implementations can be verified. In principle, it could also be implemented directly, so that the specification could simply be executed. The power of the propagator language is demonstrated by showing how a variety of common algorithms and techniques (and some less common ones) can be expressed in the language.

The second contribution is a family of measures, *Path Entropy* and *Motif Entropy*,

for measuring the degree of order inherent in coloured graphs. These generalise the standard concept of *Image Entropy*, which is insensitive to patterns and restricted to a rectangular lattice. *Path Entropy* applies broadly in any setting where a graph representation can be used and captures what is intuitively perceived as "pattern", for at least some kinds of patterns of interest in morphology. There is reason to believe that this information-theoretic measure is generally useful. The evidence for this is that, when applied to the output of models of epithelial tissues, there is good agreement with laboratory experiment. This measure then provides support for the claim that the model is a good one and purports to explain the underlying dynamics that give rise to the observed pattern.

The next contribution is the generalisation of *Path Entropy* to the *Structural Entropy* of continuous geometrical objects. This adds a spatial character to the measure. Importantly, this is a contribution to an open question in anatomy. It has long been known that structure emerges as organisms develop. This may be obvious—from an initial mass of cells comes differentiation and spatial arrangement to form complex yet well-ordered organ systems. There has been little study before now of how to quantify this phenomenon, and the relationship between ill-defined notions of complexity and order have remained muddy. *Structural Entropy* provides a way to think about the emergence of order in biological organisms in a systematic way.

The second half of this thesis contains a series of contributions related to synthetic genetic circuits. Chapter 5 describes a method for annotating rules used for describing interactions in molecular biology. Crucially, this method is backwards-compatible: it does not require any modification to the interpreter of the rule-language. An insight arises from this work: that the distinction, which at first seems to be well-defined, between annotation and the object to be annotated, is in fact dependent on the intended use of the annotations and objects. This idea itself is not new, it was previously described, for example, by Buneman et al. (2013). The application of relativity of annotation in the domain of rule-based models is new, and affords a new way of looking at rules: as being annotations describing relations among agents. This phenomenon was demonstrated to good effect by showing how to use annotations to generate contact map visualisations of the interactions between agents.

In Chapter 6 the contributions are a language for describing and a compiler for generating genetic circuits. The input to the compiler is posed purely in the annotation language and the objects of annotation are generated. The objects are absent from the input for good reason: by changing the context in which the annotations

are interpreted, it is possible to produce different kinds of objects. It is possible to produce rules for simulation in any number of rule languages. Whereas programs in rule-based languages are considered *information resources*, that is, they exist as files or sequences of bytes, it is equally possible to interpret the circuit description as referring to *non-information resources* such as actual genetic parts which can be assembled in the laboratory. The Genetic Circuit Description Language (GCDL) provides a sufficiently broad, implementation-independent way to describe genetic circuits in computer simulation and laboratory experiment.

The Genetic Circuit Compiler (GCC) is a tool that generates rule-based programs from GCDL descriptions. In addition to being a crucial piece of the infrastructure for working with genetic circuits, there is another contribution here that lies in its novel implementation. The GCDL is a Semantic Web language. This means that the often neglected facilities of logical inference are available, and the GCC is designed around them. A common criticism of logical inference in the Semantic Web realm is the lack of a way to account for context. The GCC demonstrates how contextual reasoning with Semantic Web data can be accomplished. Context is used to determine how the input is to be interpreted, i.e. in terms of the required kind of output. From this demonstration the following general observation arises: inference rules can be understood as determining the meaning of terms and the choice of rules as determining context. This contribution is important both as a demonstration of the use of contextual reasoning in the Semantic Web and for the general observation that results.

The final contribution addresses the question of how to find synthetic genetic circuits that meet a certain specification. Circuits are typically designed by a combination of trial and error and/or educated guesswork. Chapter 7 adds a new technique to the toolbox of synthetic genetic circuit design. Taking inspiration from nature, which has, after all, evolved many complex genomes adapted to the needs of biological organisms, the technique uses an evolutionary algorithm to evolve, in simulation, synthetic circuits from a library of building blocks. There are several sub-problems to be solved to make this work. An important one is to define a framework for evaluating the fitness or performance of a candidate circuit. Accomplishing this returns us to the central thread of measurement.

There is significant scope for future research suggested by the work described in this thesis. Future research directions are highlighted in the concluding sections of each chapter, fitting their stand-alone character, yet it is useful to highlight some of them here.

Already mentioned is the possibility of a direct implementation of the propagator language. Such a project would provide two-fold benefit. First it would enable a kind of practice where, when a model is described in a research paper, today usually in an under-specified way, it could be accompanied by a succinct and accurate description in the propagator language. The question of how efficient such an implementation could be remains open. Even if it is not possible to make the implementation as efficient as coding it in a low-level language, the propagator description could function as a canonical reference to verify that other implementations actually represent the behaviour of the same model. Secondly it would be useful for teaching and research in numerical methods. Propagators are already used to describe certain kinds of numerical integrators. The propagator language generalises these and so opens up the possibility of testing new kinds of integrators. Being able to write an integrator in a way that corresponds directly to the mathematical description, and then to execute it would be a useful tool for in this domain.

In Chapter 3, the claim was made that the vertex models of epithelial tissues like that of Farhadifar et al. (2007) capture the kind of mechanical dynamics that reproduce the laboratory results of Cachat et al. (2016). An interesting question is whether there are simpler models which also capture these dynamics. A promising line of inquiry has been suggested by Sachs and Danos (private correspondence) involving treating cells as freely moving point masses with interactions depending on their type. In a sense this model is simpler, however it may lack the realism required to be convincing to a biologist. Nevertheless, we can apply the measurement tools described here to evaluate the extent to which models such as that produce the kinds of patterning that is observed in the laboratory.

The observation that the rate of pattern formation, as measured by Path Entropy, depends on the underlying physical properties of the tissue suggests a strategy for parameter fitting. Vertex models can display a variety of behaviours from quasi-liquid to completely static depending on the parameters representing edge elasticity and perimeter contractility. The standard method for estimating these values for a physical tissue is laser ablation, which is rather destructive. Finding parameter values that faithfully represent a given physical tissue in a model is not straightforward, especially if the goal is to do so without destroying the cells in the process. Exhaustively searching the parameter space is very computationally intensive so, in order to constrain the search space, a library of simulations, characterised by Path Entropy, its time derivative, and other measures could be created. Since these measures can be evaluated against an

image of the physical tissue as it develops, they could be used to look up the model parameters that produce similar behaviour. Developing such a library of tissues and their measures, and correctly calibrating against measurements taken using the destructive technique, is a potential future research thread.

A pervasive shortcoming exposed by this work is lack of data. Highlighted in Chapter 4 is the small amount of three dimensional (or even two dimensional) anatomical data. Though the results reported here with the application of Structural Entropy to the data that we have are promising, further validation based on more data would be beneficial. If anatomical models of different organisms at different stages of development were available there may well be significant differences in the rate and manner in which order emerges and these differences might be made plain using a measure such as Structural Entropy. Such work must wait for the availability of suitable datasets.

Chapters 5 through 7 suggest an entire research programme dedicated to putting a synthetic genetic circuit development pipeline into practice. Such a pipeline, consisting of simulation activities leading to laboratory production of engineered cells and validation, is still quite far off. Genetic part databases, the basic requirement, still contain insufficient information (particularly about interaction rates) to serve as input for accurate simulation. It is not feasible, in general, to accurately estimate interaction rates from the underlying chemistry, and so they must be measured in the laboratory. This measurement is technically challenging and the number of parts that are available for *in vitro* assembly is large. There is also no standard, as yet, for communicating genetic circuit specifications to the assembly facilities. It is possible that the Genetic Circuit Description Language (GCDL) could form the basis for such a standard and developing it to the point of standardisation is another thread of future work. Additionally, the main mechanism of genetic circuit behaviour that has been considered here is protein-DNA-RNA interaction. There are, of course, other mechanisms. Future research could extend both the GCDL and the Genetic Circuit Compiler (GCC) to include computations with protein-protein interactions (Gao et al., 2018), Non-Ribosomal Peptide Synthases (NRPS) (Cane et al., 1998; Fischbach et al., 2006; Strieker et al., 2010), CRISPR/Cas systems (Cong et al., 2013) that modify the genetic material itself, synthetic guanine exchange factors (Yeh et al., 2007) and possibly others.

The terrain that we have covered in this thesis is varied. We have progressed from a theoretical framework for representing computational models, through the mechanics of cellular monolayers, information-theoretic image processing and solid geometry, annotation and knowledge representation, logical inference with the semantic

web, genetic circuit design and evolutionary algorithms. Most of this work has an interdisciplinary character, bringing concepts from mathematics and computer science to biology and anatomy. Though there might be some novelty in their formulation—Path Entropy as a generalisation of Image Entropy to arbitrary coloured graphs is, I believe, a new idea—the majority of these concepts are hardly at the very forefront of mathematics. Setting up a random walk through a partitioned three dimensional space might seem like a fairly obvious thing to do. But what is not obvious is that doing this leads to a way of understanding what has long been known by anatomists but has not been rigorously articulated previously. Similarly, the concept of annotation has been studied as an entity in its own right, but its application, with an understanding of its subtleties, to synthetic genetic circuits turns out to be very productive.

There are very many concepts in mathematics and computer science that have yet to find purchase beyond those fields. They remain as beautiful and fascinating pieces of abstract art, appreciated by those who know and understand them. Too often boundaries between disciplines become barriers that prevent theoretical concepts from finding productive concrete applications and at the same time rob us of ways of understanding the world around ourselves. In the course of producing this body of work, it is my hope to have bridged a small number of those barriers.

Bibliography

ACUFF, Richard, 1988. *KSL Lisp Environment Requirements* (cit. on p. 93).

ALTUN, Z.F.; HERNDON, L.A.; WOLKOW, C.A.; CROCKER, C.; LINTS, R.; HALL, D.H., 2002-2018. *WormAtlas*. Available also from: http://www.wormatlas.org (cit. on p. 63).

ANDRIANANTOANDRO, Ernesto; BASU, Subhayu; KARIG, David K; WEISS, Ron, 2006. Synthetic biology: new engineering rules for an emerging discipline. *Molecular systems biology*. Vol. 2, no. 1. Available from DOI: 10.1038/msb4100073 (cit. on p. 158).

ARMIT, Chris; BALDOCK, Richard; BRUNE, Renske; BURTON, Nick; DAVIDSON, Duncan; DELAURIER, April; GRAHAM, Elizabeth; HILL, Bill; JACOBS, Russell; LAWSON, Kirstie; LOGAN, Malcolm; MOHUN, Tim; MOSS, Julie; RICHARDSON, Lorna; RUFFINS, Seth, 2017. *3D Embryo Models and Anatomy Delineations* [https://datashare.is.ed.ac.uk/handle/10283/2805]. Edinburgh DataShare. Available also from: https://datashare.is.ed.ac.uk/handle/10283/2805 (cit. on p. 71).

ARMIT, Chris; RICHARDSON, Lorna; HILL, Bill; YANG, Yiya; BALDOCK, Richard A, 2015. eMouseAtlas informatics: embryo atlas and gene expression database. *Mammalian Genome*. Vol. 26, no. 9-10, pp. 431–440 (cit. on p. 63).

ARMIT, Chris; VENKATARAMAN, Shanmugasundaram; RICHARDSON, Lorna; STEVENSON, Peter; MOSS, Julie; GRAHAM, Liz; ROSS, Allyson; YANG, Yiya; BURTON, Nicholas; RAO, Jianguo, et al., 2012. eMouseAtlas, EMAGE, and the spatial dimension of the transcriptome. *Mammalian genome*. Vol. 23, no. 9-10, pp. 514–524 (cit. on p. 63).

ARNOLD, Vladimir Igorevich, 1992. *Ordinary Differential Equations*. Springer Verlag. Springer Textbook. ISBN 9783540548133. Available also from: https://books.google.co.uk/books?id=JUoyqlW7PZgC (cit. on p. 12).

ARONOFF, David M; OATES, John A; BOUTAUD, Olivier, 2006. New insights into the mechanism of action of acetaminophen: Its clinical pharmacologic characteristics reflect its inhibition of the two prostaglandin H2 synthases. *Clinical Pharmacology & Therapeutics*. Vol. 79, no. 1, pp. 9–19 (cit. on p. 2).

ASHBURNER, John; FRISTON, Karl J, 2000. Voxel-based morphometry—the methods. *Neuroimage*. Vol. 11, no. 6, pp. 805–821 (cit. on p. 78).

ASHBURNER, Michael; BALL, Catherine A.; BLAKE, Judith A.; BOTSTEIN, David; BUTLER, Heather; CHERRY, J. Michael; DAVIS, Allan P.; DOLINSKI, Kara; DWIGHT, Selina S.; EPPIG, Janan T.; HARRIS, Midori A.; HILL, David P.; ISSEL-TARVER, Laurie; KASARSKIS, Andrew; LEWIS, Suzanna; MATESE, John C.; RICHARDSON, Joel E.; RINGWALD, Martin; RUBIN, Gerald M.; SHERLOCK, Gavin, 2000. Gene Ontology: tool for the unification of biology. *Nature Genetics*. Vol. 25, no. 1, pp. 25–29. ISSN 1061-4036. Available from DOI: 10.1038/75556 (cit. on p. 113).

BAEZ, John C.; BIAMONTE, Jacob, 2012. A Course on Quantum Techniques for Stochastic Mechanics. *arXiv preprint*. Available from arXiv: 1209.3632 [math.NA, quant.PH]. arXiv: 1209.3632 (cit. on pp. 12, 26, 28).

BAIROCH, Amos, 2000. The ENZYME database in 2000. *Nucleic Acids Research*. Vol. 28, pp. 304–305. Available from DOI: 10.1093/nar/28.1.304 (cit. on p. 92).

BALÁZSI, Gábor; OUDENAARDEN, Alexander van; COLLINS, James J, 2011. Cellular decision making and biological noise: from microbes to mammals. *Cell*. Vol. 144, no. 6, pp. 910–925. Available from DOI: 10.1016/j.cell.2011.01.030 (cit. on p. 138).

BALDOCK, Richard A; BARD, Jonathan BL; BURGER, Albert; BURTON, Nicolas; CHRISTIANSEN, Jeff; FENG, Guanjie; HILL, Bill; HOUGHTON, Derek; KAUF-MAN, Matthew; RAO, Jianguo, et al., 2003. Emap and emage. *Neuroinformatics*. Vol. 1, no. 4, pp. 309–325 (cit. on p. 63).

BALDWIN, Geoff, 2012. *Synthetic biology: A primer*. John Wiley & Sons (cit. on pp. 108, 137, 141).

BARNES, Nick, 2010. Publish your computer code: it is good enough. *Nature News*. Vol. 467, no. 7317, pp. 753–753 (cit. on p. 10).

BEAL, Jacob; PHILLIPS, Andrew; DENSMORE, Douglas; CAI, Yizhi, 2011. High-Level Programming Languages for Biomolecular Systems. In: *Design and Analysis of Biomolecular Circuits: Engineering Approaches to Systems and Synthetic Biology*. Ed. by KOEPPL, Heinz; SETTI, Gianluca; BERNARDO, Mario di; DENSMORE, Douglas. New York, NY: Springer New York, pp. 225–252. ISBN 978-1-4419-6766-4. Available from DOI: 10.1007/978-1-4419-6766-4_11 (cit. on p. 108).

BECKETT, Dave, 2015. *Redland RDF libraries* [http://librdf.org/]. Available also from: http://librdf.org (cit. on p. 99).

BEHR, Nicolas; DANOS, Vincent; GARNIER, Ilias, 2016. Stochastic mechanics of graph rewriting. In: *Stochastic mechanics of graph rewriting. Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 46–55 (cit. on p. 12).

BEHR, Nicolas; DUCHAMP, Gérard HE; PENSON, Karol A, 2017. Combinatorics of chemical reaction systems. *arXiv preprint arXiv:1712.06575*. Available from arXiv: 1712.06575 (cit. on p. 26).

BERNERS-LEE, Tim, 2005. *An RDF language for the Semantic Web*. Available also from: https://www.w3.org/DesignIssues/Notation3. Design Issues. World Wide Web Consortium (cit. on pp. 109, 114).

BERNERS-LEE, Tim; CONNOLLY, Dan, 2011. *Notation3 (N3): A readable RDF syntax*. Available also from: https://www.w3.org/TeamSubmission/n3/. W3C Team Submission. World Wide Web Consortium (cit. on pp. 113, 129).

BERNERS-LEE, Tim; CONNOLLY, Dan; KAGAL, Lalana; SCHARF, Yosi; HENDLER, Jim, 2008. N3logic: A logical framework for the world wide web. *Theory and Practice of Logic Programming*. Vol. 8, no. 03, pp. 249–269 (cit. on pp. 113, 129).

BLANES, Sergio; CASAS, Fernando; MURUA, Ander, 2006. Composition methods for differential equations with processing. *SIAM Journal on Scientific Computing*. Vol. 27, no. 6, pp. 1817–1843. Available from DOI: 10.1137/030601223 (cit. on pp. 11, 22).

BLANES, Sergio; CASAS, Fernando; MURUA, Ander, 2008. Splitting and composition methods in the numerical integration of differential equations. *arXiv preprint*. Available from arXiv: 0812.0377 [math.NA] (cit. on p. 22).

BLASS, Andreas; DERSHOWITZ, Nachum; GUREVICH, Yuri, 2009. When are two algorithms the same? *Bulletin of Symbolic Logic*. Vol. 15, no. 2, pp. 145–168 (cit. on p. 10).

BLINOV, M L; RUEBENACKER, O; MORARU, I I, 2008. Complexity and modularity of intracellular networks: a systematic approach for modelling and simulation. *Systems Biology, IET*. Vol. 2, pp. 363–368. ISSN 1751-8849. Available from DOI: 10.1049/iet-syb:20080092 (cit. on pp. 99, 102, 138).

BLINOV, Michael L; FAEDER, James R; GOLDSTEIN, Byron; HLAVACEK, William S, 2004. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*. Vol. 20, no. 17, pp. 3289–3291 (cit. on pp. 7, 109, 138).

BLINOV, Michael L; RUEBENACKER, Oliver; SCHAFF, JamesC; MORARU, IonI, 2010. Modeling without Borders: Creating and Annotating VCell Models Using the Web. In: BORODOVSKY, Mark; GOGARTEN, JohannPeter; PRZYTYCKA, TeresaM; RAJASEKARAN, Sanguthevar (eds.). *Bioinformatics Research and Applications*. Springer Verlag, vol. 6053, pp. 3–17. Lecture Notes in Computer Science. ISBN 978-3-642-13077-9. Available from DOI: 10.1007/978-3-642-13078-6_3 (cit. on pp. 82, 103).

BONGERS, Roger S.; VEENING, Jan-Willem; WIERINGEN, Maarten Van; KUIPERS, Oscar P.; KLEEREBEZEM, Michiel, 2005. Development and Characterization of a Subtilin-Regulated Expression System in Bacillus subtilis: Strict Control of Gene Expression by Addition of Subtilin. *Applied and Environmental Microbiology* [online]. Vol. 71, no. 12, pp. 8818–8824 [visited on 2017-01-28]. ISSN 0099-2240, 1098-5336. ISSN 0099-2240, 1098-5336. Available from DOI: 10.1128/AEM.71.12.8818-8824.2005 (cit. on p. 121).

BOTTING, Regina; AYOUB, Samir S, 2005. COX-3 and the mechanism of action of paracetamol/acetaminophen. *Prostaglandins, Leukotrienes and Essential Fatty Acids*. Vol. 72, no. 2, pp. 85–87 (cit. on p. 2).

BOTTOU, Léon, 1998. Online Algorithms and Stochastic Approximations. In: SAAD, David (ed.). *Online Learning and Neural Networks*. Cambridge, UK: Cambridge University Press. Available also from: http://leon.bottou.org/papers/bottou-98x. revised, oct 2012 (cit. on p. 25).

BOUTILLIER, Pierre; MAASHA, Mutaamba; LI, Xing; MEDINA-ABARCA, Héctor F; KRIVINE, Jean; FERET, Jérôme; CRISTESCU, Ioana; FORBES, Angus G; FONTANA, Walter, 2018. The Kappa platform for rule-based modeling. *Bioinformatics*. Vol. 34, no. 13, pp. i583–i592. Available from DOI: 10.1093/bioinformatics/bty272 (cit. on pp. 7, 138).

BRICKLEY, Dan; GUHA, Ramanathan V, 2014. *RDF Schema 1.1*. Available also from: https://www.w3.org/TR/rdf-schema/. W3C Recommendation. World Wide Web Consortium (cit. on pp. 113, 115).

BROYDEN, Charles George, 1967. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*. Vol. 21, no. 99, pp. 368–381 (cit. on pp. 21, 25).

BROYDEN, Charles George, 1970a. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*. Vol. 6, no. 1, pp. 76–90. Available from DOI: doi:10.1093/imamat/6.1.76 (cit. on p. 25).

BROYDEN, Charles George, 1970b. The convergence of a class of double-rank minimization algorithms: 2. The new algorithm. *IMA journal of applied mathematics*. Vol. 6, no. 3, pp. 222–231. Available from DOI: 10.1093/imamat/6.3.222 (cit. on p. 25).

BUCHER, Dominik; HONORATO-ZIMMER, Ricardo; WAITES, William, 2014. *Module Integration Simulator*. Available also from: https://github.com/edinburgh-rbm/mois (cit. on p. 18).

BUCHLER, Nicolas E; GERLAND, Ulrich; HWA, Terence, 2003. On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences*. Vol. 100, no. 9, pp. 5136–5141 (cit. on pp. 6, 138).

BUNEMAN, Peter; KOSTYLEV, Egor V; VANSUMMEREN, Stijn, 2013. Annotations are relative. In: *Annotations are relative. Proceedings of the 16th International Conference on Database Theory*, pp. 177–188. ISBN 978-1-4503-1598-2. Available from DOI: 10.1145/2448496.2448518 (cit. on pp. 6, 85, 87, 164).

BUNKE, Horst; SHEARER, Kim, 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*. Vol. 19, no. 3, pp. 255–259 (cit. on p. 41).

BURRAGE, Kevin; HEGLAND, MARKUS; MACNAMARA, Shev; SIDJE, Roger, et al., 2006. A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In: *A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. Proc. of The AA Markov 150th Anniversary Meeting*. No. 21-37 (cit. on p. 29).

CACHAT, Élise; LIU, Weijia; MARTIN, Kim C.; YUAN, Xiaofei; YIN, Huabing; HOHENSTEIN, Peter; DAVIES, Jamie A., 2016. 2- and 3-dimensional synthetic large-scale de novo patterning by mammalian cells through phase separation. *Scientific Reports*. Vol. 6, pp. 20664. ISSN 2045-2322. Available from DOI: 10.1038/srep20664 (cit. on pp. 4, 40, 50, 166).

CAHN, John W; HILLIARD, John E, 1958. Free energy of a nonuniform system. I. Interfacial free energy. *The Journal of chemical physics*. Vol. 28, no. 2, pp. 258–267 (cit. on p. 40).

CAI, Yizhi; BEAL, Jacob; PHILLIPS, Andrew; DENSMORE, Douglas, 2011. High-level programming language for Bio-molecular systems. In: *Design and Analysis of Biomolecular Circuits*. SpringerLink, pp. 225–252 (cit. on p. 108).

CAI, Yizhi; HARTNETT, Brian; GUSTAFSSON, Claes; PECCOUD, Jean, 2007. A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. *Bioinformatics*. Vol. 23, no. 20, pp. 2760–2767. ISSN 1367-4803, 1460-2059. ISSN 1367-4803, 1460-2059. Available from DOI: 10.1093/bioinformatics/btm446 (cit. on pp. 132 sq., 139, 142).

CANE, David E; WALSH, Christopher T; KHOSLA, Chaitan, 1998. Harnessing the biosynthetic code: combinations, permutations, and mutations. *Science*. Vol. 282, no. 5386, pp. 63–68. Available from DOI: 10.1126/science.282.5386.63 (cit. on pp. 161, 167).

CANTON, Barry; LABNO, Anna; ENDY, Drew, 2008. Refinement and standardization of synthetic biological parts and devices. *Nature biotechnology*. Vol. 26, no. 7, pp. 787 (cit. on pp. 6, 138).

CAO, Youfang; TEREBUS, Anna; LIANG, Jie, 2016. State space truncation with quantified errors for accurate solutions to discrete chemical master equation. *Bulletin of mathematical biology*. Vol. 78, no. 4, pp. 617–661 (cit. on p. 29).

CAVALIERE, Matteo; DANOS, Vincent; HONORATO-ZIMMER, Ricardo; WAITES, William, 2019. Annotations for Rule-Based Models. In: *Modeling Biomolecular Site Dynamics: Methods and Protocols*. Ed. by HLAVACEK, William S. New York, NY: Springer New York, pp. 271–296. ISBN 978-1-4939-9102-0. Available from DOI: 10.1007/978-1-4939-9102-0_13 (cit. on pp. 5, 138).

CHAN, Micaela Y; PARK, Denise C; SAVALIA, Neil K; PETERSEN, Steven E; WIG, Gagan S, 2014. Decreased segregation of brain systems across the healthy adult lifespan. *Proceedings of the National Academy of Sciences*. Vol. 111, no. 46, pp. E4997–E5006 (cit. on p. 64).

CHAO, Ran; MISHRA, Shekhar; SI, Tong; ZHAO, Huimin, 2017. Engineering biological systems using automated biofoundries. *Metabolic engineering*. Vol. 42, pp. 98–108. Available from DOI: 10.1016/j.ymben.2017.06.003 (cit. on pp. 139, 161).

CHICONE, Carmen; LATUSHKIN, Yuri, 1999. *Evolution semigroups in dynamical systems and differential equations*. American Mathematical Soc. No. 70 (cit. on p. 10).

CHRISTIANSEN, Jeffrey H; YANG, Yiya; VENKATARAMAN, Shanmugasundaram; RICHARDSON, Lorna; STEVENSON, Peter; BURTON, Nicholas; BALDOCK, Richard A; DAVIDSON, Duncan R, 2006. EMAGE: a spatial database of gene expression patterns during mouse embryo development. *Nucleic acids research*. Vol. 34, no. suppl_1, pp. D637–D641 (cit. on p. 63).

CHYLEK, Lily A; HARRIS, Leonard A; TUNG, Chang-Shung; FAEDER, James R; LOPEZ, Carlos F; HLAVACEK, William S, 2014. Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*. Vol. 6, pp. 13–36. ISSN 1939-005X. Available from DOI: 10.1002/wsbm.1245 (cit. on p. 138).

CHYLEK, Lily A; HU, Bin; BLINOV, Michael L; EMONET, Thierry; FAEDER, James R; GOLDSTEIN, Byron; GUTENKUNST, Ryan N; HAUGH, Jason M; LIPNIACKI, Tomasz; POSNER, Richard G; YANG, Jin; HLAVACEK, William S, 2011. Guidelines for visualizing and annotating rule-based models. *Molecular BioSystems*. Vol. 7, pp. 2779–2795. ISSN 1742-206X. Available from DOI: 10.1039/c1mb05077j (cit. on pp. 82, 102).

CONG, Le; RAN, F Ann; COX, David; LIN, Shuailiang; BARRETTO, Robert; HABIB, Naomi; HSU, Patrick D; WU, Xuebing; JIANG, Wenyan; MARRAFFINI, Luciano, et al., 2013. Multiplex genome engineering using CRISPR/Cas systems. *Science*, pp. 1231143. Available from DOI: 10.1126/science.1231143 (cit. on pp. 161, 167).

CONN, Andrew R; GOULD, Nicholas IM; TOINT, Ph L, 1991. Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical programming*. Vol. 50, no. 1-3, pp. 177–195 (cit. on p. 25).

CONSORTIUM, UniProt et al., 2008. The universal protein resource (UniProt). *Nucleic acids research*. Vol. 36, no. suppl 1, pp. D190–D195 (cit. on pp. 113, 119).

COOLING, M. T.; ROUILLY, V.; MISIRLI, G.; LAWSON, J.; YU, T.; HALLINAN, J.; WIPAT, A., 2010. Standard virtual biological parts: a repository of modular modeling components for synthetic biology. *Bioinformatics*. Vol. 26, no. 7, pp. 925–931. ISSN 1367-4803, 1460-2059. ISSN 1367-4803, 1460-2059. Available from DOI: 10.1093/bioinformatics/btq063 (cit. on pp. 6 sq., 102, 112 sq.).

COURTOT, Mélanie; JUTY, Nick; KNÜPFER, Christian; WALTEMATH, Dagmar; ZHUKOVA, Anna; DRÄGER, Andreas; DUMONTIER, Michel; FINNEY, Andrew; GOLEBIEWSKI, Martin; HASTINGS, Janna; HOOPS, Stefan; KEATING, Sarah; KELL, Douglas B; KERRIEN, Samuel; LAWSON, James; LISTER, Allyson; LU, James; MACHNE, Rainer; MENDES, Pedro; POCOCK, Matthew; RODRIGUEZ, Nicolas; VILLEGER, Alice; WILKINSON, Darren J; WIMALARATNE, Sarala; LAIBE, Camille; HUCKA, Michael; LE NOVÈRE, Nicolas, 2011. Controlled vocabularies and semantics in systems biology. *Molecular Systems Biology*. Vol. 7. Available also from: http://msb.embopress.org/content/7/1/543.abstract (cit. on pp. 92, 197).

COX, Robert Sidney; SURETTE, Michael G; ELOWITZ, Michael B, 2007. Programming gene expression with combinatorial promoters. *Mol. Syst. Biol.* Vol. 3, no. 1, pp. 145. ISSN 1744-4292. (cit. on p. 119).

CRESS, Brady F; TOPARLAK, Ö Duhan; GULERIA, Sanjay; LEBOVICH, Matthew; STIEGLITZ, Jessica T; ENGLAENDER, Jacob A; JONES, J Andrew; LINHARDT, Robert J; KOFFAS, Mattheos AG, 2015. CRISPathBrick: modular combinatorial assembly of type II-A CRISPR arrays for dCas9-mediated multiplex transcriptional repression in E. coli. *ACS synthetic biology*. Vol. 4, no. 9, pp. 987–1000 (cit. on p. 134).

CROCE, Carlo M, 2008. Oncogenes and cancer. *New England Journal of Medicine*. Vol. 358, no. 5, pp. 502–511 (cit. on p. 59).

CUELLAR, Autumn A; LLOYD, Catherine M; NIELSEN, Poul F; BULLIVANT, David P; NICKERSON, David P; HUNTER, Peter J, 2003. An Overview of CellML 1.1, a Biological Model Description Language. *SIMULATION*. Vol. 79, no. 12, pp. 740–747. Available from DOI: 10.1177/0037549703040939 (cit. on p. 93).

CUTURI, Marco; DOUCET, Arnaud, 2014. Fast computation of Wasserstein barycenters. In: *Fast computation of Wasserstein barycenters*. *International Conference on Machine Learning*, pp. 685–693 (cit. on p. 41).

CYGANIAK, Richard; CARROLL, Jeremy; LANTHALER, Marcus, 2014. *RDF 1.1 Concepts and Abstract Syntax*. Available also from: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. W3C Recommendation. World Wide Web Consortium (cit. on pp. 91, 109, 113, 115, 132, 143 sq.).

DANOS, Vincent; FERET, Jérôme; FONTANA, Walter; HARMER, Russ; KRIVINE, Jean, 2009. Rule-Based Modelling and Model Perturbation. In: PRIAMI, Corrado; BACK, Ralph-Johan; PETRE, Ion (eds.). *Transactions on Computational Systems Biology XI*. Springer Verlag, vol. 5750, pp. 116–137. Lecture Notes in Computer Science. ISBN 978-3-642-04185-3. Available from DOI: 10.1007/978-3-642-04186-0_6 (cit. on p. 99).

DANOS, Vincent; FERET, Jérôme; FONTANA, Walter; HARMER, Russell; KRIVINE, Jean, 2007. Rule-Based Modelling of Cellular Signalling. In: CAIRES, Luís; VASCONCELOS, Vasco T. (eds.). *CONCUR 2007 – Concurrency Theory*. Springer Verlag, pp. 17–41. Lecture Notes in Computer Science, no. 4703. ISBN 978-3-540-74407-8. Available from DOI: 10.1007/978-3-540-74407-8_3 (cit. on pp. 6, 109).

DANOS, Vincent; FERET, Jérôme; FONTANA, Walter; HARMER, Russell; KRIVINE, Jean, 2008. Rule-based modelling, symmetries, refinements. In: *Rule-based modelling, symmetries, refinements*. *Formal methods in systems biology*. Springer Verlag, pp. 103–122. ISBN 978-3-540-68413-8. Available from DOI: 10.1007/978-3-540-68413-8_8 (cit. on pp. 6, 111, 138).

DANOS, Vincent; FERET, Jérôme; FONTANA, Walter; KRIVINE, Jean, 2007. Scalable Simulation of Cellular Signaling Networks. In: SHAO, Zhong (ed.). *APLAS*. Springer Verlag, vol. 4807, pp. 139–157. Lecture Notes in Computer Science. ISBN 978-3-540-76636-0. Available from DOI: 10.1007/978-3-540-76637-7_10 (cit. on p. 82).

DANOS, Vincent; LANEVE, Cosimo, 2004. Formal molecular biology. *Theoretical Computer Science*. Vol. 325, pp. 69–110. ISSN 0304-3975. Available from DOI: 10.1016/j.tcs.2004.03.065 (cit. on pp. 82, 99, 111, 138).

DAVIDSON, Duncan; BARD, Jonathan; KAUFMAN, Matthew; BALDOCK, Richard, 2001. The Mouse Atlas Database: a community resource for mouse development. *TRENDS in Genetics*. Vol. 17, no. 1, pp. 49–51 (cit. on p. 63).

DAVIES, Jamie A, 2016. Anatomical complexity is acquired at an exponential rate during mouse embryonic development. *Matters*. Vol. 2, no. 4, pp. e201604000005 (cit. on pp. 5, 64, 78).

DAVIES, Jamie A.; LAWRENCE, Melanie L., 2018. *Organoids and mini-organs*. Elsevier. Available from DOI: 10.1016/B978-0-12-812636-3.00015-8 (cit. on p. 79).

DCMI USAGE BOARD, 2012. *DCMI Metadata Terms*. Available also from: http://www.dublincore.org/documents/dcmi-terms (cit. on p. 91).

DE JONG, Hidde, 2002. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*. Vol. 9, no. 1, pp. 67–103 (cit. on p. 138).

DEGTYARENKO, Kirill; MATOS, Paula de; ENNIS, Marcus; HASTINGS, Janna; ZBINDEN, Martin; MCNAUGHT, Alan; ALCÁNTARA, Rafael; DARSOW, Michael; GUEDJ, Mickaël; ASHBURNER, Michael, 2008. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research*. Vol. 36, no. suppl 1, pp. D344–D350. Available from DOI: 10.1093/nar/gkm791 (cit. on pp. 92, 197).

DEHMER, Matthias; MOWSHOWITZ, Abbe, 2011. A history of graph entropy measures. *Information Sciences*. Vol. 181, no. 1, pp. 57–78. ISSN 0020-0255. Available from DOI: 10.1016/j.ins.2010.08.041 (cit. on p. 45).

DEL VECCHIO, Domitilla; NINFA, Alexander J; SONTAG, Eduardo D, 2008. Modular cell biology: retroactivity and insulation. *Molecular systems biology*. Vol. 4, no. 1, pp. 161 (cit. on pp. 6, 138).

DEMIR, Emek; CARY, Michael P; PALEY, Suzanne; FUKUDA, Ken; LEMER, Christian; VASTRIK, Imre; WU, Guanming; D'EUSTACHIO, Peter; SCHAEFER, Carl; LUCIANO, Joanne; SCHACHERER, Frank; MARTINEZ-FLORES, Irma; HU, Zhenjun; JIMENEZ-JACINTO, Veronica; JOSHI-TOPE, Geeta; KANDASAMY, Kumaran; LOPEZ-FUENTES, Alejandra C; MI, Huaiyu; PICHLER, Elgar; RODCHENKOV, Igor; SPLENDIANI, Andrea; TKACHEV, Sasha; ZUCKER, Jeremy; GOPINATH, Gopal; RAJASIMHA, Harsha; RAMAKRISHNAN, Ranjani; SHAH, Imran; SYED, Mustafa; ANWAR, Nadia; BABUR, Ozgun; BLINOV, Michael; BRAUNER, Erik; CORWIN, Dan; DONALDSON, Sylva; GIBBONS, Frank; GOLDBERG, Robert; HORNBECK, Peter; LUNA, Augustin; MURRAY-RUST, Peter; NEUMANN, Eric; REUBENACKER, Oliver; SAMWALD, Matthias; IERSEL, Martijn van; WIMALARATNE, Sarala; ALLEN, Keith; BRAUN, Burk; WHIRL-CARRILLO, Michelle; CHEUNG, Kei-Hoi; DAHLQUIST, Kam; FINNEY, Andrew; GILLESPIE, Marc; GLASS, Elizabeth; GONG, Li; HAW, Robin; HONIG, Michael; HUBAUT, Olivier; KANE, David; KRUPA, Shiva; KUTMON, Martina; LEONARD, Julie; MARKS, Debbie; MERBERG, David; PETRI, Victoria; PICO, Alex; RAVENSCROFT, Dean; REN, Liya; SHAH, Nigam; SUNSHINE, Margot; TANG, Rebecca; WHALEY, Ryan; LETOVKSY, Stan; BUETOW, Kenneth H;

RZHETSKY, Andrey; SCHACHTER, Vincent; SOBRAL, Bruno S; DOGRUSOZ, Ugur; MCWEENEY, Shannon; ALADJEM, Mirit; BIRNEY, Ewan; COLLADO-VIDES, Julio; GOTO, Susumu; HUCKA, Michael; NOVERE, Nicolas Le; MALTSEV, Natalia; PANDEY, Akhilesh; THOMAS, Paul; WINGENDER, Edgar; KARP, Peter D; SANDER, Chris; BADER, Gary D, 2010. The BioPAX community standard for pathway data sharing. *Nat Biotech*. Vol. 28, no. 9, pp. 935–942. ISBN 1087-0156. Available from DOI: 10.1038/nbt.1666 (cit. on pp. 92, 197).

DRUMMOND, Nick; SHEARER, Rob, 2006. The open world assumption. In: *The open world assumption. eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web*. Vol. 15 (cit. on p. 114).

EILBECK, Karen; LEWIS, Suzanna; MUNGALL, Christopher; YANDELL, Mark; STEIN, Lincoln; DURBIN, Richard; ASHBURNER, Michael, 2005. The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biology*. Vol. 6, no. 5, pp. R44. ISBN 1465-6906. Available also from: http://genomebiology.com/2005/6/5/R44 (cit. on pp. 92, 197).

ELLSON, John; GANSNER, Emden; KOUTSOFIOS, Lefteris; NORTH, Stephen C; WOODHULL, Gordon, 2002. Graphviz—open source graph drawing tools. In: *Graphviz—open source graph drawing tools. Graph Drawing*, pp. 483–484 (cit. on p. 99).

ELOWITZ, Michael B.; LEIBLER, Stanislas, 2000. A synthetic oscillatory network of transcriptional regulators. *Nature* [online]. Vol. 403, no. 6767, pp. 335–338 [visited on 2017-01-29]. ISSN 0028-0836. Available from DOI: 10.1038/35002125 (cit. on p. 108).

ENDLER, Lukas; RODRIGUEZ, Nicolas; JUTY, Nick; CHELLIAH, Vijayalakshmi; LAIBE, Camille; LI, Chen; LE NOVÈRE, Nicolas, 2009. Designing and encoding models for synthetic biology. *Journal of The Royal Society Interface*, pp. rsif.2009.0035.focus. Available from DOI: 10.1098/rsif.2009.0035.focus (cit. on pp. 81, 93).

ENDY, Drew, 2005. Foundations for engineering biology. *Nature*. Vol. 438, no. 7067, pp. 449 (cit. on p. 138).

ESCUDERO, Luis M; COSTA, Luciano da F; KICHEVA, Anna; BRISCOE, James; FREEMAN, Matthew; BABU, M Madan, 2011. Epithelial organisation revealed by a network of cellular contacts. *Nature communications*. Vol. 2, pp. 526 (cit. on p. 41).

ESHERA, M A; FU, King-Sun, 1984. A graph distance measure for image analysis. *IEEE transactions on systems, man, and cybernetics*, pp. 398–408 (cit. on p. 41).

ESTEVA, Andre; KUPREL, Brett; NOVOA, Roberto A; KO, Justin; SWETTER, Susan M; BLAU, Helen M; THRUN, Sebastian, 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*. Vol. 542, no. 7639, pp. 115 (cit. on p. 58).

FAEDER, James R.; BLINOV, Michael L.; HLAVACEK, William S., 2009. Rule-Based Modeling of Biochemical Systems with BioNetGen. In: MALY, Ivan V (ed.). *Systems Biology*. Humana Press, vol. 500, pp. 113–167. Methods in Molecular Biology. ISBN 978-1-934115-64-0. Available from DOI: 10.1007/978-1-59745-525-1_5 (cit. on pp. 7, 82, 138).

FAN, Yiming; ZENG, Ling-Li; SHEN, Hui; QIN, Jian; LI, Fuquan; HU, Dewen, 2017. Lifespan Development of the Human Brain Revealed by Large-Scale Network Eigen-Entropy. *Entropy*. Vol. 19, no. 9, pp. 471 (cit. on p. 63).

FARHADIFAR, Reza; RÖPER, Jens-Christian; AIGOUY, Benoit; EATON, Suzanne; JÜLICHER, Frank, 2007. The Influence of Cell Mechanics, Cell-Cell Interactions, and Proliferation on Epithelial Packing. *Current Biology* [online]. Vol. 17, no. 24, pp. 2095–2104 [visited on 2015-04-30]. ISSN 0960-9822. Available from DOI: 10.1016/j.cub.2007.11.049 (cit. on pp. 3 sq., 9, 23, 35, 40, 50, 166).

FENG, Song; OLLIVIER, Julien F; SWAIN, Peter S; SOYER, Orkun S, 2015. BioJazz: in silico evolution of cellular networks with unbounded complexity using rule-based modeling. *Nucleic acids research*. Vol. 43, no. 19, pp. e123–e123 (cit. on p. 139).

FERET, Jérôme; KRIVINE, Jean, 2015. *Personal correspondence* (cit. on p. 131).

FERRY, Michael S; HASTY, Jeff; COOKSON, Natalie A, 2012. Synthetic biology approaches to biofuel production. *Biofuels*. Vol. 3, no. 1, pp. 9–12. Available from DOI: 10.4155/bfs.11.151 (cit. on pp. 108, 137).

FISCHBACH, Michael A; WALSH, Christopher T, 2006. Assembly-line enzymology for polyketide and nonribosomal peptide antibiotics: logic, machinery, and mechanisms. *Chemical reviews*. Vol. 106, no. 8, pp. 3468–3496. Available from DOI: 10.1021/cr0503097 (cit. on pp. 161, 167).

FLETCHER, Roger, 1970. A new approach to variable metric algorithms. *The computer journal*. Vol. 13, no. 3, pp. 317–322. Available from DOI: 10.1093/comjnl/13.3.317 (cit. on p. 25).

FOREST, Etienne, 2006. Geometric integration for particle accelerators. *Journal of Physics A: Mathematical and General*. Vol. 39, no. 19, pp. 5321. ISSN 0305-4470. Available also from: http://iopscience.iop.org/0305-4470/39/19/S03 (cit. on p. 11).

FORREST, Peter, 2016. The Identity of Indiscernibles. In: ZALTA, Edward N. (ed.). *The Stanford Encyclopedia of Philosophy* [https://plato.stanford.edu/archives/win2016/entries/identity-indiscernible/]. Winter 2016. Metaphysics Research Lab, Stanford University (cit. on p. 119).

FRANÇOIS, Paul; HAKIM, Vincent, 2004. Design of genetic networks with specified functions by evolution in silico. *Proceedings of the National Academy of Sciences*. Vol. 101, no. 2, pp. 580–585 (cit. on p. 139).

FRANÇOIS, Paul; SIGGIA, Eric D, 2008. A case study of evolutionary computation of biochemical adaptation. *Physical biology*. Vol. 5, no. 2, pp. 026009 (cit. on p. 139).

FUCHS, Laszlo, 2011. *Partially ordered algebraic systems*. Courier Corporation (cit. on p. 11).

FUNAHASHI, Akira; JOURAKU, Akiya; MATSUOKA, Yukiko; KITANO, Hiroaki, 2007. Integration of CellDesigner and SABIO-RK. *In Silico Biology*. Vol. 7, pp. 81–90. Available also from: http://iospress.metapress.com/content/G36020GV8774R214 (cit. on p. 82).

GALANIE, Stephanie; THODEY, Kate; TRENCHARD, Isis J.; FILSINGER INTERRANTE, Maria; SMOLKE, Christina D., 2015. Complete biosynthesis of opioids in yeast. *Science*. Vol. 349, no. 6252, pp. 1095–1100. ISSN 0036-8075. Available from DOI: 10.1126/science.aac9373 (cit. on pp. 108, 137).

GALDZICKI, Michal; CLANCY, Kevin P; OBERORTNER, Ernst; POCOCK, Matthew; QUINN, Jacqueline Y; RODRIGUEZ, Cesar A; ROEHNER, Nicholas; WILSON, Mandy L; ADAM, Laura; ANDERSON, J Christopher, et al., 2014. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology*. Vol. 32, no. 6, pp. 545–550. Available from DOI: 10.1038/nbt.2891 (cit. on p. 109).

GALDZICKI, Michal; CLANCY, Kevin P; OBERORTNER, Ernst; POCOCK, Matthew; QUINN, Jacqueline; RODRIGUEZ, Cesar A; ROEHNER, Nicholas; WILSON, Mandy L; ADAM, Laura; ANDERSON, J Christopher; BARTLEY, Bryan A; BEAL, Jacob; CHANDRAN, Deepak; CHEN, Joanna; DENSMORE, Douglas; ENDY, Drew; GRÜNBERG, Raik; HALLINAN, Jennifer; HILLSON, Nathan J; JOHNSON, Jeffrey D; KUCHINSKY, Allan; LUX, Matthew; MISIRLI, Göksel; PECCOUD, Jean; PLAHAR, Hector A; SIRIN, Evren; STAN, Guy-Bart; VILLALOBOS, Alan; WIPAT, Anil; GENNARI, John H; MYERS, Chris J; SAURO, Herbert M, 2014. SBOL: A community standard for communicating designs in synthetic biology. *Nature Biotechnology* (cit. on pp. 93, 138, 197).

GALDZICKI, Michal; RODRIGUEZ, Cesar; CHANDRAN, Deepak; SAURO, Herbert M; GENNARI, John H, 2011. Standard biological parts knowledgebase. *PloS one*. Vol. 6, no. 2, pp. e17005 (cit. on pp. 6, 138).

GALDZICKI, Michal; WILSON, Mandy L; RODRIGUEZ, Cesar A; POCOCK, Mathew R; OBERORTNER, Ernst; ADAM, Laura; ADLER, Aaron; ANDERSON, J Christopher; BEAL, Jacob; CAI, Yizhi; CHANDRAN, Deepak; DENSMORE, Douglas; DRORY, Omri A; ENDY, Drew; GENNARI, John H; GRÜNBERG, Raik; HAM, Timothy S; HILLSON, Nathan J; JOHNSON, Jeffrey D; KUCHINSKY, Allan; LUX, Matthew W; MADSEN, Curtis; MISIRLI, Göksel; MYERS, Chris J; OLGUIN, Carlos; PECCOUD, Jean; PLAHAR, Hector; PLATT, Darren; ROEHNER, Nicholas; SIRIN, Evren; SMITH, Trevor F; STAN, Guy-Bart; VILLALOBOS, Alan; WIPAT, Anil; SAURO, Herbert M, 2012. *BBF RFC #87*. Synthetic Biology Open Language (SBOL) Version 1.1.0. Available from DOI: http://hdl.handle.net/1721.1/73909 (cit. on pp. 93, 197).

GANDON, Fabien; SCHREIBER, Guus, 2014. *RDF 1.1 XML Syntax* (cit. on p. 91).

GAO, Xiaojing J; CHONG, Lucy S; KIM, Matthew S; ELOWITZ, Michael B, 2018. Programmable protein circuits in living cells. *Science*. Vol. 361, no. 6408, pp. 1252–1258. Available from DOI: 10.1126/science.aat5062 (cit. on pp. 161, 167).

GARDNER, Timothy S; CANTOR, Charles R; COLLINS, James J, 2000. Construction of a genetic toggle switch in Escherichia coli. *Nature*. Vol. 403, no. 6767, pp. 339 (cit. on p. 149).

GHANEM, Carolina I; PÉREZ, Maríéa J; MANAUTOU, José E; MOTTINO, Aldo D, 2016. Acetaminophen from liver to brain: new insights into drug pharmacological action and toxicity. *Pharmacological research*. Vol. 109, pp. 119–131 (cit. on p. 2).

GIERER, Alfred; MEINHARDT, Hans, 1972. A theory of biological pattern formation. *Biological Cybernetics*. Vol. 12, no. 1, pp. 30–39 (cit. on p. 40).

GIVENS, Clark R.; SHORTT, Rae Michael, 1984. A class of Wasserstein metrics for probability distributions. *The Michigan Mathematical Journal*. Vol. 31, no. 2, pp. 231–240. ISSN 0026-2285, 1945-2365. ISSN 0026-2285, 1945-2365. Available from DOI: 10.1307/mmj/1029003026 (cit. on p. 160).

GOLDFARB, Donald, 1970. A family of variable-metric methods derived by variational means. *Mathematics of computation*. Vol. 24, no. 109, pp. 23–26. Available from DOI: 10.1090/S0025-5718-1970-0258249-6 (cit. on p. 25).

GOLDSTEIN, Herbert, 1980. *Classical Mechanics*. 2nd ed. Addison-Wesley. ISBN 9780201029185 (cit. on pp. 20, 23).

GOLUB, Gene H; VAN LOAN, Charles F, 2012. *Matrix computations*. JHU Press (cit. on p. 32).

GONZALEZ, Rafael C; EDDINS, Steven L; WOODS, Richard E, 2004. *Digital Image Publishing Using MATLAB*. Prentice Hall (cit. on pp. 41, 66).

GOOD, Catriona D; JOHNSRUDE, Ingrid S; ASHBURNER, John; HENSON, Richard NA; FRISTON, Karl J; FRACKOWIAK, Richard SJ, 2001. A voxel-based morphometric study of ageing in 465 normal adult human brains. *Neuroimage*. Vol. 14, no. 1, pp. 21–36 (cit. on p. 79).

GRANER, François; GLAZIER, James A., 1992. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical Review Letters*. Vol. 69, no. 13, pp. 2013–2016. Available from DOI: 10.1103/PhysRevLett.69.2013 (cit. on p. 40).

GRIZZI, Fabio; CHIRIVA-INTERNATI, Maurizio, 2005. The complexity of anatomical systems. *Theoretical Biology and Medical Modelling*. Vol. 2, no. 1, pp. 26 (cit. on pp. 5, 62).

GUET, Călin C; ELOWITZ, Michael B; HSING, Weihong; LEIBLER, Stanislas, 2002. Combinatorial synthesis of genetic networks. *Science*. Vol. 296, no. 5572, pp. 1466–1470 (cit. on pp. 134, 139, 156).

GUILLAUD, Martial; CLEM, Carole; MACAULAY, Calum, 2010. An in silico platform for the study of epithelial pre-invasive neoplastic development. *Biosystems*. Vol. 102, no. 1, pp. 22–31 (cit. on p. 59).

HAIRER, Ernst; WANNER, Gerhard, 1996. *Solving Ordinary Differential Equations II*. 2nd ed. Springer Verlag. Springer Series in Computational Mathematics, no. 14. ISBN 978-3-642-05220-0, 978-3-642-05221-7. Available also from: http://link.springer.com/book/10.1007/978-3-642-05221-7 (cit. on pp. 11, 20 sqq.).

HALLINAN, Jennifer; GILFELLON, Owen; MISRILI, Göksel; WIPAT, Anil, 2014. Tuning receiver characteristics in bacterial quorum communication: An evolutionary approach using standard virtual biological parts. In: *Tuning receiver characteristics in bacterial quorum communication: An evolutionary approach using standard virtual biological parts. 2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pp. 1–8. Available from DOI: 10.1109/CIBCB.2014.6845520 (cit. on p. 109).

HALPIN, Harry; HAYES, Patrick J.; MCCUSKER, James P.; MCGUINNESS, Deborah L.; THOMPSON, Henry S., 2010. When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data. In: PATEL-SCHNEIDER, Peter F.; PAN, Yue; HITZLER, Pascal; MIKA, Peter; ZHANG, Lei; PAN, Jeff Z.; HORROCKS, Ian; GLIMM, Birte (eds.). *The Semantic Web – ISWC 2010* [online]. Springer Verlag, pp. 305–320 [visited on 2017-01-29]. Lecture Notes in Computer Science. ISBN 978-3-642-17745-3 978-3-642-17746-0. Available from: http://link.springer.com/chapter/10.1007/978-3-642-17746-0_20. DOI: 10.1007/978-3-642-17746-0_20 (cit. on pp. 114, 119).

HARMELEN, Frank van; MCGUINNESS, Deborah L, 2004. *OWL Web Ontology Language*. Available also from: http://www.w3.org/TR/2004/REC-owl-features-20040210/. W3C Recommendation. World Wide Web Consortium} (cit. on pp. 91, 113).

HARRIS, L. A.; HOGG, J. S.; TAPIA, J.-J.; SEKAR, J. A. P.; KORSUNSKY, I.; ARORA, A.; BARUA, D.; SHEEHAN, R. P.; FAEDER, J. R., 2015. BioNetGen 2.2: Advances in Rule-Based Modeling. *ArXiv e-prints*. Available from arXiv: 1507.03572 [q-bio.QM] (cit. on p. 102).

HARRIS, Leonard A.; HOGG, Justin S.; TAPIA, José-Juan; SEKAR, John A. P.; GUPTA, Sanjana; KORSUNSKY, Ilya; ARORA, Arshi; BARUA, Dipak; SHEEHAN, Robert P.; FAEDER, James R., 2016. BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics*. Vol. 32, no. 21, pp. 3366–3368. ISSN 1367-4803. Available from DOI: 10.1093/bioinformatics/btw469 (cit. on p. 109).

HEATH, Tom; BIZER, Christian, 2011. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*. Vol. 1, no. 1, pp. 1–136. Available from DOI: 10.2200/S00334ED1V01Y201102WBE001 (cit. on p. 7).

HEDLEY, W J; NELSON, M R; BELLIVANT, D P; NIELSEN, P F, 2001. A short introduction to CellML. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*. Vol. 359, pp. 1073–1089. Available from DOI: 10.1098/rsta.2001.0817 (cit. on p. 93).

HÉNON, Michel, 1976. A two-dimensional mapping with a strange attractor. In: *A two-dimensional mapping with a strange attractor. The Theory of Chaotic Attractors*. Springer Verlag, pp. 94–102 (cit. on p. 18).

HILL, Bill; BALDOCK, Richard A, 2015. Constrained distance transforms for spatial atlas registration. *BMC bioinformatics*. Vol. 16, no. 1, pp. 90 (cit. on p. 71).

HLAVACEK, William S; FAEDER, James R; BLINOV, Michael L; PERELSON, Alan S; GOLDSTEIN, Byron, 2003. The complexity of complexes in signal transduction. *Biotechnology and bioengineering*. Vol. 84, no. 7, pp. 783–794 (cit. on p. 111).

HOCHBRUCK, M.; LUBICH, C.; SELHOFER, H., 1998. Exponential Integrators for Large Systems of Differential Equations. *SIAM Journal on Scientific Computing*. Vol. 19, no. 5, pp. 1552–1574. ISSN 1064-8275. Available from DOI: 10.1137/S1064827595295337 (cit. on p. 11).

HORN, Andreas; OSTWALD, Dirk; REISERT, Marco; BLANKENBURG, Felix, 2014. The structural–functional connectome and the default mode network of the human brain. *Neuroimage*. Vol. 102, pp. 142–151 (cit. on p. 63).

HORROCKS, Ian, 2005. OWL: A Description Logic Based Ontology Language. In: OWL. *Logic Programming* [online]. Springer, Berlin, Heidelberg, pp. 1–4 [visited on 2017-03-14]. Available from DOI: 10.1007/11562931_1 (cit. on p. 113).

HUANG, Hao; ZHANG, Jiangyang; WAKANA, Setsu; ZHANG, Weihong; REN, Tianbo; RICHARDS, Linda J; YAROWSKY, Paul; DONOHUE, Pamela; GRAHAM, Ernest; ZIJL, Peter CM van, et al., 2006. White and gray matter development in human fetal, newborn and pediatric brains. *Neuroimage*. Vol. 33, no. 1, pp. 27–38 (cit. on p. 64).

HUCKA, Michael; FINNEY, Andrew; SAURO, Herbert M; BOLOURI, Hamid; DOYLE, John C; KITANO, Hiroaki; ARKIN, Adam P; BORNSTEIN, Benjamin J; BRAY, Dennis; CORNISH-BOWDEN, Athel, et al., 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*. Vol. 19, pp. 524–531. Available from DOI: 10.1093/bioinformatics/btg015 (cit. on p. 93).

HYDE, Lewis, 2008. *Trickster makes this world: How disruptive imagination creates culture*. Canongate Books (cit. on p. 1).

HYLAND, Bernadette; ATEMEZING, Ghislain; VILLAZÓN-TERRAZAS, Boris, 2014. *Best Practices for Publishing Linked Data*. Available also from: https://www.w3.org/TR/ld-bp/. W3C Working Group Note. World Wide Web Consortium (cit. on p. 113).

INCE, Darrel C; HATTON, Leslie; GRAHAM-CUMMING, John, 2012. The case for open computer programs. *Nature*. Vol. 482, no. 7386, pp. 485 (cit. on p. 10).

JAHNKE, Tobias; HUISINGA, Wilhelm, 2007. Solving the chemical master equation for monomolecular reaction systems analytically. *Journal of mathematical biology*. Vol. 54, no. 1, pp. 1–26. Available from DOI: 10.1007/s00285-006-0034-x (cit. on p. 34).

JANZEN, David; SAIEDIAN, Hossein, 2005. Test-driven development concepts, taxonomy, and future direction. *Computer*. Vol. 38, no. 9, pp. 43–50 (cit. on p. 147).

JAYNES, Edwin T, 1965. Gibbs vs Boltzmann entropies. *American Journal of Physics*. Vol. 33, no. 5, pp. 391–398 (cit. on p. 41).

JONES, Simon Peyton, 2003. *Haskell 98 language and libraries: the revised report*. Cambridge University Press (cit. on p. 14).

JUTY, Nick; LE NOVÈRE, Nicolas; LAIBE, Camille, 2012. Identifiers.org and MIRIAM Registry: community resources to provide persistent identification. *Nucleic Acids Research*. Vol. 40, no. D1, pp. D580–D586. Available from DOI: 10.1093/nar/gkr1097 (cit. on p. 92).

KANEHISA, Minoru; ARAKI, Michihiro; GOTO, Susumu; HATTORI, Masahiro; HIRAKAWA, Mika; ITOH, Masumi; KATAYAMA, Toshiaki; KAWASHIMA, Shuichi; OKUDA, Shujiro; TOKIMATSU, Toshiaki; YAMANISHI, Yoshihiro, 2008. KEGG for linking genomes to life and the environment. *Nucl. Acids Res.* Vol. 36, no. suppl_1, pp. D480–484. Available from DOI: 10.1093/nar/gkm882 (cit. on p. 92).

KARR, Jonathan R.; SANGHVI, Jayodita C.; MACKLIN, Derek N.; GUTSCHOW, Miriam V.; JACOBS, Jared M.; BOLIVAL, Benjamin; ASSAD-GARCIA, Nacyra; GLASS, John I.; COVERT, Markus W., 2012. A whole-cell computational model predicts phenotype from genotype. *Cell*. Vol. 150, no. 2, pp. 389–401. ISSN 1097-4172. Available from DOI: 10.1016/j.cell.2012.05.044 (cit. on p. 3).

KATOK, Anatole; HASSELBLATT, Boris, 1995. *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press. Encyclopedia of Mathematics and its Applications. Available from DOI: 10.1017/CBO9780511809187 (cit. on pp. 3, 10 sq.).

KAUFFMAN, Stuart A, 1993. *The origin of order*. Oxford University Press New York (cit. on pp. 5, 62).

KITTLESON, Joshua T; WU, Gabriel C; ANDERSON, J Christopher, 2012. Successes and failures in modular genetic engineering. *Current opinion in chemical biology*. Vol. 16, no. 3-4, pp. 329–336 (cit. on p. 138).

KLEMENT, M; DĚD, T; ŠAFRÁNEK, D; ČERVENÝ, J; MÜLLER, S; STEUER, R, 2014. Biochemical Space: A Framework for Systemic Annotation of Biological Models. *Electronic Notes in Theoretical Computer Science*. Vol. 306, pp. 31–44. ISSN 1571-0661. Available from DOI: http://dx.doi.org/10.1016/j.entcs.2014.06.013 (cit. on pp. 7, 82).

KÖHLER, Agnes; KRIVINE, Jean; VIDMAR, Jakob, 2014. A Rule-Based Model of Base Excision Repair. In: MENDES, Pedro; DADA, Joseph O.; SMALLBONE, Kieran (eds.). *Computational Methods in Systems Biology - 12th International Conference, CMSB 2014, Manchester, UK, November 17-19, 2014, Proceedings*. Springer Verlag, vol. 8859, pp. 173–195. Lecture Notes in Computer Science. ISBN 978-3-319-12981-5. Available from DOI: 10.1007/978-3-319-12982-2_13 (cit. on p. 82).

KRAUSE, Falko; UHLENDORF, Jannis; LUBITZ, Timo; SCHULZ, Marvin; KLIPP, Edda; LIEBERMEISTER, Wolfram, 2010. Annotation and merging of SBML models with semanticSBML. *Bioinformatics*. Vol. 26, pp. 421–422. Available from DOI: 10.1093/bioinformatics/btp642 (cit. on p. 7).

KRIVINE, Jean; FERET, Jérôme, 2018. *KaSim*. Available also from: http://www.kappalanguage.org/ (cit. on pp. 7, 93, 109, 111, 121, 140).

KULLBACK, Solomon; LEIBLER, Richard A., 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics*. Vol. 22, no. 1, pp. 79–86. ISSN 0003-4851. Available also from: http://www.jstor.org/stable/2236703 (cit. on pp. 47, 160).

KUNZE, John A.; BAKER, Thomas, 2007. *RFC5013 – The Dublin Core Metadata Element Set* [online] [visited on 2017-01-29]. Available from: https://tools.ietf.org/html/rfc5013. Request for Comments. Internet Engineering Task Force (cit. on p. 117).

LARKIN, Kieran G., 2016. Reflections on Shannon Information: In search of a natural information-entropy for images. *CoRR*. Vol. abs/1609.01117. Available from arXiv: 1609.01117 (cit. on pp. 41, 44).

LARSON, Anne M; POLSON, Julie; FONTANA, Robert J; DAVERN, Timothy J; LALANI, Ezmina; HYNAN, Linda S; REISCH, Joan S; SCHIØDT, Frank V; OSTAPOWICZ, George; SHAKIL, A Obaid, et al., 2005. Acetaminophen-induced acute liver failure: results of a United States multicenter, prospective study. *Hepatology*. Vol. 42, no. 6, pp. 1364–1372. Available from DOI: 10.1002/hep.20948 (cit. on p. 2).

LASALLE, Joseph P, 1976. *The stability of dynamical systems*. Siam (cit. on p. 10).

LE NOVÈRE, Nicolas; FINNEY, Andrew, 2005. *A simple scheme for annotating SBML with references to controlled vocabularies and database entries*. Available also from: http://www.ebi.ac.uk/compneur-srv/sbml/proposals/AnnotationURI.pdf (cit. on p. 91).

LE NOVÈRE, Nicolas; FINNEY, Andrew; HUCKA, Michael; BHALLA, Upinder S; CAMPAGNE, Fabien; COLLADO-VIDES, Julio; CRAMPIN, Edmund J; HALSTEAD, Matt; KLIPP, Edda; MENDES, Pedro; NIELSEN, Poul; SAURO, Herbert; SHAPIRO, Bruce; SNOEP, Jacky L; SPENCE, Hugh D; WANNER, Barry L, 2005. Minimum information requested in the annotation of biochemical models (MIRIAM). *Nat Biotech*. Vol. 23, no. 12, pp. 1509–1515. Available also from: http://dx.doi.org/10.1038/nbt1156 (cit. on p. 90).

LEFEVRE, James G.; CHIU, Han S.; COMBES, Alexander N.; VANSLAMBROUCK, Jessica M.; JU, Ali; HAMILTON, Nicholas A.; LITTLE, Melissa H., 2017. Self-organisation after embryonic kidney dissociation is driven via selective adhesion of ureteric epithelial cells. *Development*. Vol. 144, no. 6, pp. 1087–1096. ISSN 0950-1991, 1477-9129. ISSN 0950-1991, 1477-9129. Available from DOI: 10.1242/dev.140228 (cit. on p. 40).

LI, Chen; DONIZELLI, Marco; RODRIGUEZ, Nicolas; DHARURI, Harish; ENDLER, Lukas; CHELLIAH, Vijayalakshmi; LI, Lu; HE, Enuo; HENRY, Arnaud; STEFAN, Melanie; SNOEP, Jacky; HUCKA, Michael; LE NOVERE, Nicolas; LAIBE, Camille, 2010. BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*. Vol. 4, pp. 92. ISSN 1752-0509. Available also from: http://www.biomedcentral.com/1752-0509/4/92 (cit. on pp. 7, 81 sq., 91, 138, 197).

LISTER, Allyson L; POCOCK, Matthew; TASCHUK, Morgan; WIPAT, Anil, 2009. Saint: a lightweight integration environment for model annotation. *Bioinformatics*. Vol. 25, pp. 3026–3027. Available from DOI: 10.1093/bioinformatics/btp523 (cit. on pp. 99, 103).

LLUFRIU, Sara; MARTINEZ-HERAS, Eloy; SOLANA, Elisabeth; SOLA-VALLS, Nuria; SEPULVEDA, Maria; BLANCO, Yolanda; MARTINEZ-LAPISCINA, Elena H; ANDORRA, Magi; VILLOSLADA, Pablo; PRATS-GALINO, Alberto, et al., 2017. Structural networks involved in attention and executive functions in multiple sclerosis. *NeuroImage: Clinical*. Vol. 13, pp. 288–296 (cit. on p. 78).

LOPEZ, Carlos F; MUHLICH, Jeremy L; BACHMAN, John A; SORGER, Peter K, 2013a. Programming biological models in Python using PySB. *Molecular Systems Biology*. Vol. 9. ISSN 1744-4292. Available from DOI: 10.1038/msb.2013.1 (cit. on pp. 82, 102).

LOPEZ, Carlos F; MUHLICH, Jeremy L; BACHMAN, John A; SORGER, Peter K, 2013b. Programming biological models in Python using PySB. *Molecular Systems Biology*. Vol. 9, no. 1. ISSN 1744-4292. Available from DOI: 10.1038/msb.2013.1 (cit. on p. 138).

MACIA, Javier; SOLE, Ricard, 2014. How to make a synthetic multicellular computer. *PLoS One*. Vol. 9, no. 2, pp. e81248. Available from DOI: 10.1371/journal.pone.0081248 (cit. on p. 137).

MADSEN, Curtis; MCLAUGHLIN, James Alastair; MISIRLI, Göksel; POCOCK, Matthew; FLANAGAN, Keith; HALLINAN, Jennifer; WIPAT, Anil, 2016. The SBOL Stack: A Platform for Storing, Publishing, and Sharing Synthetic Biology Designs. *ACS Synthetic Biology*. Vol. 5, no. 6, pp. 487–497. Available from DOI: 10.1021/acssynbio.5b00210 (cit. on pp. 138, 140).

MAGNO, Ramiro; GRIENEISEN, Verônica A; MARÉE, Athanasius F M, 2015. The biophysical nature of cells: potential cell behaviours revealed by analytical and computational studies of cell surface mechanics. *BMC Biophysics*. Vol. 8, no. 1, pp. 8. ISSN 2046-1682. Available from DOI: 10.1186/s13628-015-0022-x (cit. on pp. 40, 58).

MAGRANE, Michele; UNIPROT CONSORTIUM, 2011. UniProt Knowledgebase: a hub of integrated protein data. *Database*. Vol. 2011. Available from DOI: 10.1093/database/bar009 (cit. on pp. 92, 197).

MAGUIRE, Eleanor A; GADIAN, David G; JOHNSRUDE, Ingrid S; GOOD, Catriona D; ASHBURNER, John; FRACKOWIAK, Richard SJ; FRITH, Christopher D, 2000. Navigation-related structural change in the hippocampi of taxi drivers. *Proceedings of the National Academy of Sciences*. Vol. 97, no. 8, pp. 4398–4403 (cit. on p. 79).

MANGIN, J-F, 2000. Entropy minimization for automatic correction of intensity nonuniformity. In: *Entropy minimization for automatic correction of intensity nonuniformity. Mathematical methods in biomedical image analysis, 2000. proceedings. ieee workshop on*, pp. 162–169 (cit. on pp. 41, 66).

MASINTER, Larry; BERNERS-LEE, Tim; FIELDING, Roy T., 2005. *RFC3896 – Uniform Resource Identifier (URI): Generic Syntax* [online] [visited on 2017-01-30]. Available from: https://tools.ietf.org/html/rfc3986. Request for Comments. Internet Engineering Task Force (cit. on pp. 113 sq.).

MATSUDA, Hiroshi, 2013. Voxel-based morphometry of brain MRI in normal aging and Alzheimer's disease. *Aging and disease*. Vol. 4, no. 1, pp. 29 (cit. on pp. 64, 79).

MCBEATH, Jim, 2008. *Understanding Monads*. Available also from: https://jim-mcbeath.blogspot.com/2008/12/understanding-monads.html (cit. on p. 14).

MCLACHLAN, R., 1995. On the Numerical Integration of Ordinary Differential Equations by Symmetric Composition Methods. *SIAM Journal on Scientific Computing*. Vol. 16, no. 1, pp. 151–168. ISSN 1064-8275. Available from DOI: 10.1137/0916010 (cit. on pp. 11, 22).

MENZELLA, Hugo G; REID, Ralph; CARNEY, John R; CHANDRAN, Sunil S; REISINGER, Sarah J; PATEL, Kedar G; HOPWOOD, David A; SANTI, Daniel V, 2005. Combinatorial polyketide biosynthesis by de novo design and rearrangement of modular polyketide synthase genes. *Nature biotechnology*. Vol. 23, no. 9, pp. 1171 (cit. on p. 134).

MIETCHEN, Daniel; GASER, Christian, 2009. Computational morphometry for detecting changes in brain structure due to development, aging, learning, disease and evolution. *Frontiers in neuroinformatics*. Vol. 3, pp. 25 (cit. on p. 64).

MILES, Alistair; MATTHEWS, Brian; WILSON, Michael; BRICKLEY, Dan, 2005. SKOS Core: Simple knowledge organisation for the Web. *International Conference on Dublin Core and Metadata Applications* [online]. Vol. 0, no. 0, pp. 3–10 [visited on 2017-01-29]. ISSN 1939-1366. Available from: http://dcpapers.dublincore.org/pubs/article/view/798 (cit. on pp. 115, 118).

MILO, Ron; JORGENSEN, Paul; MORAN, Uri; WEBER, Griffin; SPRINGER, Michael, 2009. BioNumbers—the database of key numbers in molecular and cell biology. *Nucleic acids research*. Vol. 38, no. suppl_1, pp. D750–D753 (cit. on p. 1).

MIRAMS, Gary R.; ARTHURS, Christopher J.; BERNABEU, Miguel O.; BORDAS, Rafel; COOPER, Jonathan; CORRIAS, Alberto; DAVIT, Yohan; DUNN, Sara-Jane; FLETCHER, Alexander G.; HARVEY, Daniel G., et al., 2013. Chaste: an open source C++ library for computational physiology and biology. *PLoS Comput Biol*. Vol. 9, no. 3, pp. e1002970 (cit. on pp. 37, 50).

MISIRLI, Göksel; CAVALIERE, Matteo; WAITES, William; POCOCK, Matthew; MADSEN, Curtis; GILFELLON, Owen; HONORATO-ZIMMER, Ricardo; ZULIANI, Paolo; DANOS, Vincent; WIPAT, Anil, 2015. Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization. *Bioinformatics*, pp. btv660. ISSN 1367-4803, 1460-2059. ISSN 1367-4803, 1460-2059. Available from DOI: 10.1093/bioinformatics/btv660 (cit. on pp. 6, 82, 85, 90, 93 sq., 96, 102, 105, 115, 119, 133, 138 sq., 143).

MISIRLI, Göksel; HALLINAN, Jennifer S; YU, Tommy; LAWSON, James R; WIMALARATNE, Sarala M; COOLING, Michael T; WIPAT, Anil, 2011. Model annotation for synthetic biology: automating model to nucleotide sequence conversion. *Bioinformatics*. Vol. 27, pp. 973–979. Available from DOI: 10.1093/bioinformatics/btr048 (cit. on pp. 102, 133).

MISIRLI, Göksel; HALLINAN, Jennifer; WIPAT, Anil, 2014. Composable Modular Models for Synthetic Biology. *J. Emerg. Technol. Comput. Syst.* [online]. Vol. 11, no. 3, pp. 22:1–22:19 [visited on 2017-01-28]. ISSN 1550-4832. Available from DOI: 10.1145/2631921 (cit. on pp. 7, 81, 102, 112 sq., 121, 138).

MISIRLI, Göksel; WAITES, William; CAVALIERE, Matteo; ZULIANI, Paolo; DANOS, Vincent; WIPAT, Anil; HONORATO-ZIMMER, Ricardo, 2016. *Modular Composition of Synthetic Biology Designs using Rule-Based Models*. Poster at IWBDA 2016 (cit. on p. 103).

MOGGI, Eugenio, 1989. Computational lambda-calculus and monads. In: *Computational lambda-calculus and monads. [1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pp. 14–23 (cit. on p. 14).

MOGGI, Eugenio, 1990. *An abstract view of programming languages*. Technical report. University of Edinburgh. Department of Computer Science. Laboratory for Foundations of Computer Science. (cit. on p. 14).

MOLER, C.; VAN LOAN, C., 2003. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Review*. Vol. 45, no. 1, pp. 3–49. ISSN 0036-1445. Available from DOI: 10.1137/S00361445024180 (cit. on pp. 12, 32).

MONTECCHI-PALAZZI, Luisa; BEAVIS, Ron; BINZ, Pierre-Alain; CHALKLEY, Robert J; COTTRELL, John; CREASY, David; SHOFSTAHL, Jim; SEYMOUR, Sean L; GARAVELLI, John S, 2008. The PSI-MOD community standard for representation of protein modification data. *Nat Biotech*. Vol. 26, pp. 864–866. ISSN 1087-0156. Available also from: http://dx.doi.org/10.1038/nbt0808-864 (cit. on pp. 98, 197).

MORARU, I I; SCHAFF, J C; SLEPCHENKO, B M; BLINOV, M L; MORGAN, F; LAKSHMINARAYANA, A; GAO, F; LI, Y; LOEW, L M, 2008. Virtual cell modelling and simulation software environment. *Systems Biology, IET*. Vol. 2, pp. 352–362. ISSN 1751-8849. Available from DOI: 10.1049/iet-syb:20080102 (cit. on pp. 7, 81, 93, 138).

MUKHERJI, Shankar; VAN OUDENAARDEN, Alexander, 2009. Synthetic biology: understanding biological design from synthetic circuits. *Nature Reviews Genetics*. Vol. 10, no. 12, pp. 859 (cit. on p. 137).

MULDER, Nicola J.; APWEILER, Rolf, 2008. The InterPro Database and Tools for Protein Domain Analysis. In: *Current Protocols in Bioinformatics*. John Wiley and Sons, Inc., pp. 2–7. ISBN 9780471250951. Available from DOI: 10.1002/0471250953.bi0207s21 (cit. on p. 197).

MUNSKY, Brian; KHAMMASH, Mustafa, 2006. The finite state projection algorithm for the solution of the chemical master equation. *The Journal of chemical physics*. Vol. 124, no. 4, pp. 044104 (cit. on p. 29).

NAGAI, Tatsuzo; HONDA, Hisao, 2001. A dynamic cell model for the formation of epithelial tissues. *Philosophical Magazine Part B*. Vol. 81, no. 7, pp. 699–719. ISSN 1364-2812. Available from DOI: 10.1080/13642810108205772 (cit. on pp. 23, 36, 40).

NATALE, Darren A.; ARIGHI, Cecilia N.; BARKER, Winona C.; BLAKE, Judith A.; BULT, Carol J.; CAUDY, Michael; DRABKIN, Harold J.; D'EUSTACHIO, Peter; EVSIKOV, Alexei V.; HUANG, Hongzhan; NCHOUTMBOUBE, Jules; ROBERTS, Natalia V.; SMITH, Barry; ZHANG, Jian; WU, Cathy H., 2011. The Protein Ontology: a structured representation of protein forms and complexes. *Nucleic Acids Research*. Vol. 39, no. suppl 1, pp. D539–D545. Available from DOI: 10.1093/nar/gkq907 (cit. on p. 197).

NEAL, Maxwell L; COOLING, Michael T; SMITH, Lucian P; THOMPSON, Christopher T; SAURO, Herbert M; CARLSON, Brian E; COOK, Daniel L; GENNARI, John H, 2014. A reappraisal of how to build modular, reusable models of biological systems. *PLoS computational biology*. Vol. 10, no. 10, pp. e1003849 (cit. on pp. 109, 112).

NEUMANN, J v, 1932. Proof of the quasi-ergodic hypothesis. *Proceedings of the National Academy of Sciences*. Vol. 18, no. 1, pp. 70–82 (cit. on p. 149).

NGUYEN, Tramy; ROEHNER, Nicholas; ZUNDEL, Zach; MYERS, Chris J., 2016. A Converter from the Systems Biology Markup Language to the Synthetic Biology Open Language. *ACS Synthetic Biology*. Vol. 5, no. 6, pp. 479–486. Available from DOI: 10.1021/acssynbio.5b00212. PMID: 26696234 (cit. on p. 112).

O'RAHILLY, Ronan; GARDNER, Ernest, 1975. The timing and sequence of events in the development of the limbs in the human embryo. *Anatomy and embryology*. Vol. 148, no. 1, pp. 1–23 (cit. on p. 1).

OSBORNE, James M.; FLETCHER, Alexander G.; PITT-FRANCIS, Joe M.; MAINI, Philip K.; GAVAGHAN, David J., 2017. Comparing individual-based approaches to modelling the self-organization of multicellular tissues. *PLOS Computational Biology*. Vol. 13, no. 2, pp. e1005387. ISSN 1553-7358. Available from DOI: 10.1371/journal.pcbi.1005387 (cit. on pp. 40, 50, 52).

PADDON, C. J.; WESTFALL, P. J.; PITERA, D. J.; BENJAMIN, K.; FISHER, K.; MCPHEE, D.; LEAVELL, M. D.; TAI, A.; MAIN, A.; ENG, D.; POLICHUK, D. R.; TEOH, K. H.; REED, D. W.; TREYNOR, T.; LENIHAN, J.; JIANG, H.; FLECK, M.; BAJAD, S.; DANG, G.; DENGROVE, D.; DIOLA, D.; DORIN, G.; ELLENS, K. W.; FICKES, S.; GALAZZO, J.; GAUCHER, S. P.; GEISTLINGER, T.; HENRY, R.; HEPP, M.; HORNING, T.; IQBAL, T.; KIZER, L.; LIEU, B.; MELIS, D.; MOSS, N.; REGENTIN, R.; SECREST, S.; TSURUTA, H.; VAZQUEZ, R.; WESTBLADE, L. F.; XU, L.; YU, M.; ZHANG, Y.; ZHAO, L.; LIEVENSE, J.; COVELLO, P. S.; KEASLING, J. D.; REILING, K. K.; RENNINGER, N. S.; NEWMAN, J. D., 2013. High-level semi-synthetic production of the potent antimalarial artemisinin.

Nature. Vol. 496, no. 7446, pp. 528–532. ISBN 0028-0836. Available also from: http://dx.doi.org/10.1038/nature12051 (cit. on pp. 108, 137).

PEDERSEN, Michael; PHILLIPS, Andrew, 2009. Towards programming languages for genetic engineering of living cells. *Journal of the Royal Society Interface*. Vol. 6, no. Suppl 4, pp. S437–S450. ISSN 1742-5689. Available from DOI: 10.1098/rsif.2008.0516.focus (cit. on pp. 6, 108, 133, 139, 142).

PENG, Roger D, 2011. Reproducible research in computational science. *Science*. Vol. 334, no. 6060, pp. 1226–1227 (cit. on p. 10).

PERRY, Alistair; WEN, Wei; LORD, Anton; THALAMUTHU, Anbupalam; ROBERTS, Gloria; MITCHELL, Philip B; SACHDEV, Perminder S; BREAKSPEAR, Michael, 2015. The organisation of the elderly connectome. *Neuroimage*. Vol. 114, pp. 414–426 (cit. on p. 78).

PINSKY, Mark; KARLIN, Samuel, 2010. *An introduction to stochastic modeling*. Academic press (cit. on p. 70).

PIPER, Jim; RUTOVITZ, Denis, 1985. Data structures for image processing in a C language and Unix environment. *Pattern Recognition Letters*. Vol. 3, no. 2, pp. 119–129 (cit. on p. 71).

PRUD'HOMMEAUX, Eric; CAROTHERS, Gavin, 2014. *RDF 1.1 Turtle*. W3C Recommendation. World Wide Web Consortium (cit. on pp. 93, 114, 132, 143).

PRUD'HOMMEAUX, Eric; SEABORNE, Andy, 2013. *SPARQL Query Language for RDF* [Available at http://www.w3.org/TR/rdf-sparql-query (23/01/2015)] (cit. on p. 94).

PURNICK, Priscilla EM; WEISS, Ron, 2009. The second wave of synthetic biology: from modules to systems. *Nature reviews Molecular cell biology*. Vol. 10, no. 6, pp. 410. Available from DOI: 10.1038/nrm2698 (cit. on p. 159).

REDDICK, Wilburn E; GLASS, John O; COOK, Edwin N; ELKIN, T David; DEATON, Russell J, 1997. Automated segmentation and classification of multispectral magnetic resonance images of brain using artificial neural networks. *IEEE Transactions on medical imaging*. Vol. 16, no. 6, pp. 911–918 (cit. on p. 59).

REUS, Marcel A de; SAENGER, Victor M; KAHN, René S; HEUVEL, Martijn P van den, 2014. An edge-centric perspective on the human connectome: link communities in the brain. *Phil. Trans. R. Soc. B*. Vol. 369, no. 1653, pp. 20130527 (cit. on p. 78).

REYNOLDS, Osborne, 1883. XXIX. An experimental investigation of the circumstances which determine whether the motion of water shall he direct or sinuous, and of the law of resistance in parallel channels. *Philosophical Transactions of the Royal Society of London*. Vol. 174, pp. 935–982 (cit. on p. 23).

RICHARDSON, Lorna; VENKATARAMAN, Shanmugasundaram; STEVENSON, Peter; YANG, Yiya; BURTON, Nicholas; RAO, Jianguo; FISHER, Malcolm; BALDOCK, Richard A; DAVIDSON, Duncan R; CHRISTIANSEN, Jeffrey H, 2009. EMAGE mouse embryo spatial gene expression database: 2010 update. *Nucleic acids research*. Vol. 38, no. suppl_1, pp. D703–D709 (cit. on pp. 5, 63).

RICHARDSON, Lorna; VENKATARAMAN, Shanmugasundaram; STEVENSON, Peter; YANG, Yiya; MOSS, Julie; GRAHAM, Liz; BURTON, Nicholas; HILL, Bill; RAO, Jianguo; BALDOCK, Richard A, et al., 2013. EMAGE mouse embryo spatial gene expression database: 2014 update. *Nucleic acids research*. Vol. 42, no. D1, pp. D835–D844 (cit. on p. 63).

RIPLEY, BD, 1990. Thoughts on pseudorandom number generators. *Journal of Computational and Applied Mathematics*. Vol. 31, no. 1, pp. 153–163 (cit. on p. 13).

ROEHNER, Nicholas; ZHANG, Zhen; NGUYEN, Tramy; MYERS, Chris J., 2015. Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language. *ACS Synthetic Biology*. Vol. 4, no. 8, pp. 873–879. Available from DOI: 10.1021/sb5003289. PMID: 25822671 (cit. on p. 112).

RONACHER, Armin, 2008. *Jinja2 (The Python Template Engine)*. Available also from: http://jinja.pocoo.org (cit. on p. 130).

RUBNER, Yossi; TOMASI, Carlo; GUIBAS, Leonidas J, 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*. Vol. 40, no. 2, pp. 99–121 (cit. on p. 41).

RUDER, Warren C.; LU, Ting; COLLINS, James J., 2011. Synthetic Biology Moving into the Clinic. *Science*. Vol. 333, no. 6047, pp. 1248–1252. ISSN 0036-8075. Available from DOI: 10.1126/science.1206843 (cit. on pp. 108, 137).

RUIZ, Alberto, 2012. *Introduction to hmatrix*. Available also from: http://dis.um.es/~alberto/material/hmatrix.pdf (cit. on p. 32).

SÁNCHEZ-GUTIÉRREZ, Daniel; TOZLUOGLU, Melda; BARRY, Joseph D; PASCUAL, Alberto; MAO, Yanlan; ESCUDERO, Luis M, 2015. Fundamental physical cellular constraints drive self-organization of tissues. *The EMBO Journal*. Vol. 35, no. 1, pp. 77–88. ISSN 0261-4189. Available from DOI: 10.15252/embj.201592374 (cit. on pp. 40, 45).

SANCHEZ, Alvaro; GARCIA, Hernan G; JONES, Daniel; PHILLIPS, Rob; KONDEV, Jané, 2011. Effect of promoter architecture on the cell-to-cell variability in gene expression. *PLoS Comput. Biol.* Vol. 7, no. 3, pp. e1001100. ISSN 1553-7358. (cit. on p. 119).

SANDERSIUS, SA; WEIJER, Cornelis J; NEWMAN, Timothy J, 2011. Emergent cell and tissue dynamics from subcellular modeling of active biomechanical processes. *Physical biology*. Vol. 8, no. 4, pp. 045007 (cit. on p. 40).

SAUERMANN, Leo; CYGANIAK, Richard; VÖLKEL, Max, 2011. *Cool URIs for the semantic web* [online] [visited on 2017-03-14]. Available from DOI: 10.22028/D291-25086. Technical report. Deutsches Forschungszentrum für Künstliche Intelligenz (cit. on p. 113).

SHANNO, David F, 1970. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*. Vol. 24, no. 111, pp. 647–656. Available from DOI: 10.1090/S0025-5718-1970-0274029-X (cit. on p. 25).

SHANNON, Claude E., 2001. A Mathematical Theory of Communication. *SIGMOBILE Mob. Comput. Commun. Rev.* Vol. 5, no. 1, pp. 3–55. ISSN 1559-1662. Available from DOI: 10.1145/584091.584093 (cit. on pp. 41, 62, 160).

SHEARER, Rob; MOTIK, Boris; HORROCKS, Ian, 2008. HermiT: A Highly-Efficient OWL Reasoner. In: *HermiT: A Highly-Efficient OWL Reasoner. OWLED*. Vol. 432, p. 91 (cit. on p. 101).

SHETTY, Reshma P; ENDY, Drew; KNIGHT, Thomas F, 2008. Engineering BioBrick vectors from BioBrick parts. *Journal of biological engineering*. Vol. 2, no. 1, pp. 1 (cit. on p. 6).

SIRIN, Evren; PARSIA, Bijan; GRAU, Bernardo Cuenca; KALYANPUR, Aditya; KATZ, Yarden, 2007. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*. Vol. 5, no. 2, pp. 51–53 (cit. on p. 101).

SKILLING, John; BRYAN, RK, 1984. Maximum entropy image reconstruction: general algorithm. *Monthly notices of the royal astronomical society*. Vol. 211, no. 1, pp. 111–124 (cit. on p. 41).

SMANSKI, Michael J; BHATIA, Swapnil; ZHAO, Dehua; PARK, YongJin; WOODRUFF, Lauren BA; GIANNOUKOS, Georgia; CIULLA, Dawn; BUSBY, Michele; CALDERON, Johnathan; NICOL, Robert, 2014. Functional optimization of gene clusters by combinatorial design and assembly. *Nature biotechnology*. Vol. 32, no. 12, pp. 1241 (cit. on p. 134).

SMITH, Barry; CEUSTERS, Werner; KLAGGES, Bert; KOHLER, Jacob; KUMAR, Anand; LOMAX, Jane; MUNGALL, Chris; NEUHAUS, Fabian; RECTOR, Alan; ROSSE, Cornelius, 2005. Relations in biomedical ontologies. *Genome Biology*. Vol. 6, no. 5, pp. R46. ISBN 1465-6906. Available also from: http://genomebiology.com/2005/6/5/R46 (cit. on pp. 92, 197).

SMITH, Cyril Stanley, 1952. Grain Shapes and Other Metallurgical Applications of Topology. *Metal Interfaces* (cit. on p. 52).

SMITH, Cyril Stanley, 2015. Grain Shapes and Other Metallurgical Applications of Topology. *Metallography, Microstructure, and Analysis*. Vol. 4, no. 6, pp. 543–567. ISSN 2192-9262, 2192-9270. ISSN 2192-9262, 2192-9270. Available from DOI: 10.1007/s13632-015-0241-1 (cit. on pp. 35 sq.).

SNOEP, Jacky L; OLIVIER, Brett G, 2003. JWS Online Cellular Systems Modelling and Microbiology. *Microbiology*. Vol. 149, pp. 3045–3047. Available from DOI: 10.1099/mic.0.C0124-0 (cit. on pp. 7, 81, 138).

SOMOGYI, Roland; SNIEGOSKI, Carol Ann, 1996. Modeling the complexity of genetic networks: understanding multigenic and pleiotropic regulation. *complexity*. Vol. 1, no. 6, pp. 45–63 (cit. on p. 138).

SPORNS, Olaf; TONONI, Giulio; EDELMAN, Gerald M, 2000. Connectivity and complexity: the relationship between neuroanatomy and brain dynamics. *Neural networks*. Vol. 13, no. 8-9, pp. 909–922 (cit. on p. 63).

STALLMAN, Richard et al., 1992. *GNU coding standards* [https://www.gnu.org/prep/standards/]. Available also from: https://www.gnu.org/prep/standards/ (cit. on p. 93).

STAPLE, D. B.; FARHADIFAR, R.; RÖPER, J.-C.; AIGOUY, B.; EATON, S.; JÜLICHER, F., 2010. Mechanics and remodelling of cell packings in epithelia. *The European Physical Journal E*. Vol. 33, no. 2, pp. 117–127. ISSN 1292-8941, 1292-895X. ISSN 1292-8941, 1292-895X. Available from DOI: 10.1140/epje/i2010-10677-0 (cit. on pp. 40, 58).

STEINBERG, Malcolm S, 1962. On the mechanism of tissue reconstruction by dissociated cells, III. Free energy relations and the reorganization of fused, heteronomic tissue fragments. *Proceedings of the National Academy of Sciences*. Vol. 48, no. 10, pp. 1769–1776 (cit. on p. 39).

STICKEL, Mark E; TYSON, Mabry, 1985. An analysis of consecutively bounded depth-first search with applications in automated deduction. In: *An analysis of consecutively bounded depth-first search with applications in automated deduction. IJCAI*, pp. 1073–1075 (cit. on p. 45).

STRIEKER, Matthias; TANOVIĆ, Alan; MARAHIEL, Mohamed A, 2010. Nonribosomal peptide synthetases: structures and dynamics. *Current opinion in structural biology*. Vol. 20, no. 2, pp. 234–240. Available from DOI: 10.1016/j.sbi.2010.01.009 (cit. on pp. 161, 167).

SU, Shengran; GAO, Zhifan; ZHANG, Heye; LIN, Qiang; HAU, William Kongto; LI, Shuo, 2017. Detection of lumen and media-adventitia borders in IVUS images using sparse auto-encoder neural network. In: *Detection of lumen and media-adventitia borders in IVUS images using sparse auto-encoder neural network. Biomedical Imaging (ISBI 2017), 2017 IEEE 14th International Symposium on*, pp. 1120–1124 (cit. on p. 59).

SWAINSTON, Neil; MENDES, Pedro, 2009. libAnnotationSBML: a library for exploiting SBML annotations. *Bioinformatics*. Vol. 25, pp. 2292–2293. Available from DOI: 10.1093/bioinformatics/btp392 (cit. on p. 92).

TANGE, O., 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*. Vol. 36, no. 1, pp. 42–47. Available from DOI: http://dx.doi.org/10.5281/zenodo.16303.

TAPIA, Jose-Juan; FAEDER, James R., 2013. The Atomizer: Extracting Implicit Molecular Structure from Reaction Network Models. In: *The Atomizer: Extracting Implicit Molecular Structure from Reaction Network Models. Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*. Wshington DC, USA: ACM, 726:726–726:727. BCB'13. ISBN 978-1-4503-2434-2. Available from DOI: 10.1145/2506583.2512389 (cit. on p. 102).

TESTA, Cristina; LAAKSO, Mikko P; SABATTOLI, Francesca; ROSSI, Roberta; BELTRAMELLO, Alberto; SOININEN, Hilkka; FRISONI, Giovanni B, 2004. A comparison between the accuracy of voxel-based morphometry and hippocampal volumetry in Alzheimer's disease. *Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine*. Vol. 19, no. 3, pp. 274–282 (cit. on pp. 64, 79).

TEUKOLSKY, Saul; VETTERLING, William; FLANNERY, Brian, 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press (cit. on pp. 13, 20).

THE GENE ONTOLOGY CONSORTIUM, 2001. Creating the Gene Ontology Resource: Design and Implementation. *Genome Research*. Vol. 11, no. 8, pp. 1425–1433. Available from DOI: 10.1101/gr.180801 (cit. on pp. 92, 197).

THEILER, Karl, 1989. *The house mouse: atlas of embryonic development*. Springer Verlag. ISBN 978-3-642-88420-7 (cit. on p. 71).

TONONI, Giulio; SPORNS, Olaf; EDELMAN, Gerald M, 1994. A measure for brain complexity: relating functional segregation and integration in the nervous system. *Proceedings of the National Academy of Sciences*. Vol. 91, no. 11, pp. 5033–5037 (cit. on p. 63).

TRIBUS, Myron; MCIRVINE, Edward C, 1971. Energy and information. *Scientific American*. Vol. 225, no. 3, pp. 179–190 (cit. on p. 77).

TSAI, Du-Yih; LEE, Yongbum; MATSUYAMA, Eri, 2008. Information entropy measure for evaluation of image quality. *Journal of digital imaging*. Vol. 21, no. 3, pp. 338–347 (cit. on pp. 41, 66).

TURING, Alan M., 1952. The Chemical Basis of Morphogenesis. *Phil. Trans. R. Soc. Lond. B*. Vol. 237, pp. 37–72 (cit. on pp. 4, 40).

UNBEKANDT, Mathieu; DAVIES, Jamie A., 2010. Dissociation of embryonic kidneys followed by reaggregation allows the formation of renal tissues. *Kidney International*. Vol. 77, no. 5, pp. 407–416. ISSN 0085-2538. Available from DOI: 10.1038/ki.2009.482 (cit. on p. 40).

VAN ESSEN, David C; SMITH, Stephen M; BARCH, Deanna M; BEHRENS, Timothy EJ; YACOUB, Essa; UGURBIL, Kamil; CONSORTIUM, Wu-Minn HCP, et al., 2013. The WU-Minn human connectome project: an overview. *Neuroimage*. Vol. 80, pp. 62–79 (cit. on p. 63).

VICENTE-MUNUERA, Pablo; GOMEZ-GALVEZ, Pedro; TAGUA, Antonio; LETRAN, Marta; MAO, Yanlan; ESCUDERO, Luis M, 2017. EpiGraph: an open-source platform to quantify epithelial organization. *bioRxiv*, pp. 217521. Available also from: http://biorxiv.org/content/early/2017/11/13/217521 (cit. on pp. 40, 45).

WADLER, Philip, 1990. Comprehending monads. In: *Comprehending monads. Proceedings of the 1990 ACM conference on LISP and functional programming*, pp. 61–78 (cit. on p. 14).

WADLER, Philip, 1995. Monads for functional programming. In: *Monads for functional programming. International School on Advanced Functional Programming*, pp. 24–52 (cit. on p. 14).

WADLER, Philip, 1997. How to declare an imperative. *ACM Computing Surveys (CSUR)*. Vol. 29, no. 3, pp. 240–263 (cit. on p. 14).

WAITES, William; CAVALIERE, Matteo; CACHAT, Élise; DANOS, Vincent; DAVIES, Jamie A, 2018. An information-theoretic measure for patterning in epithelial tissues. *IEEE Access*. Vol. 6, pp. 40302–40312 (cit. on pp. 4, 37, 62, 66).

WAITES, William; DAVIES, Jamie A, 2019. Emergence of Structure in Mouse Embryos: Structural Entropy Morphometry Applied to Segmented Anatomical Models. *Journal of Anatomy*. Vol. 234, no. 4, pp. 706–715 (cit. on p. 5).

WAITES, William; MISIRLI, Göksel; CAVALIERE, Matteo; DANOS, Vincent; WIPAT, Anil, 2018. A Genetic Circuit Compiler: Generating Combinatorial Genetic Circuits with Web Semantics and Inference. *ACS Synthetic Biology*. Vol. 7, no. 12, pp. 2812–2823. Available from DOI: 10.1021/acssynbio.8b00201 (cit. on pp. 6, 139, 143, 160).

WANG, Baojun; KITNEY, Richard I; JOLY, Nicolas; BUCK, Martin, 2011. Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology. *Nature communications*. Vol. 2, pp. 508 (cit. on pp. 119, 138).

WEAIRE, Denis; RIVIER, Nicholas, 2009. Soap, cells and statistics – random patterns in two dimensions. *Contemporary Physics*. Vol. 50, no. 1, pp. 199–239. ISSN 0010-7514. Available from DOI: 10.1080/00107510902734680 (cit. on pp. 36, 52).

WILSON-KANAMORI, John; DANOS, Vincent; THOMSON, Ty; HONORATO-ZIMMER, Ricardo, 2015. Kappa Rule-Based Modeling in Synthetic Biology. In: MARCHISIO, Mario Andrea (ed.). *Computational Methods in Synthetic Biology* [online]. Springer New York, pp. 105–135 [visited on 2017-01-09]. Methods in Molecular Biology, no. 1244. ISBN 978-1-4939-1877-5. Available from: http://dx.doi.org/10.1007/978-1-4939-1878-2_6. DOI: 10.1007/978-1-4939-1878-2_6 (cit. on pp. 122, 138).

XU, Chenchu; XU, Lei; GAO, Zhifan; ZHAO, Shen; ZHANG, Heye; ZHANG, Yanping; DU, Xiuquan; ZHAO, Shu; GHISTA, Dhanjoo; LI, Shuo, 2017. Direct detection of pixel-level myocardial infarction areas via a deep-learning algorithm. In: *Direct detection of pixel-level myocardial infarction areas via a deep-learning algorithm. International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 240–249 (cit. on p. 59).

XU, Wen; SMITH, Adam M; FAEDER, James R; MARAI, G Elisabeta, 2011. RuleBender: a visual interface for rule-based modeling. *Bioinformatics*. Vol. 27, pp. 1721–1722. Available from DOI: 10.1093/bioinformatics/btr197 (cit. on p. 93).

YANOFSKY, Noson S, 2010. Towards a definition of an algorithm. *Journal of Logic and Computation*. Vol. 21, no. 2, pp. 253–286 (cit. on p. 10).

YEH, Brian J; RUTIGLIANO, Robert J; DEB, Anrica; BAR-SAGI, Dafna; LIM, Wendell A, 2007. Rewiring cellular morphology pathways with synthetic guanine nucleotide exchange factors. *Nature*. Vol. 447, no. 7144, pp. 596. Available from DOI: 10.1038/nature05851 (cit. on pp. 161, 167).

YEO, Ronald A; RYMAN, Sephira G; VAN DEN HEUVEL, Martijn P; DE REUS, Marcel A; JUNG, Rex E; POMMY, Jessica; MAYER, Andrew R; EHRLICH, Stefan; SCHULZ, S Charles; MORROW, Eric M, et al., 2016. Graph metrics of structural brain networks in individuals with schizophrenia and healthy controls: group differences, relationships with intelligence, and genetics. *Journal of the International Neuropsychological Society*. Vol. 22, no. 2, pp. 240–249 (cit. on p. 78).

YU, Kai; LIU, Chengcheng; KIM, Byung-Gee; LEE, Dong-Yup, 2015. Synthetic fusion protein design and applications. *Biotechnology advances*. Vol. 33, no. 1, pp. 155–164 (cit. on p. 121).

YU, Tommy; LLOYD, Catherine M; NICKERSON, David P; COOLING, Michael T; MILLER, Andrew K; GARNY, Alan; TERKILDSEN, Jonna R; LAWSON, James; BRITTEN, Randall D; HUNTER, Peter J; NIELSEN, Poul M F, 2011. The Physiome Model Repository 2. *Bioinformatics*. Vol. 27, pp. 743–744. Available from DOI: 10.1093/bioinformatics/btq723 (cit. on pp. 7, 81, 138).

ZHANG, Wenlu; LI, Rongjian; DENG, Houtao; WANG, Li; LIN, Weili; JI, Shuiwang; SHEN, Dinggang, 2015. Deep convolutional neural networks for multi-modality isointense infant brain image segmentation. *NeuroImage*. Vol. 108, pp. 214–224 (cit. on p. 59).

ZIV, Etay; NEMENMAN, Ilya; WIGGINS, Chris H, 2007. Optimal signal processing in small stochastic biochemical networks. *PloS one*. Vol. 2, no. 10, pp. e1077 (cit. on p. 139).

Appendix A

The Rule-Based Modelling Ontology

A.1 Conventional Namespace Prefixes

Prefixes for ontologies and controlled vocabularies used to annotate models.

Prefix	Description
rbmo	Rule-based modelling ontology (presented in this paper)
dct	Dublin Core Metadata Initiative Terms (http://www.dublincore.org/documents/dcmi-terms)
bqiol	BioModels.net Biology Qualifiers (Li et al., 2010)
go	Gene Ontology (The Gene Ontology Consortium, 2001)
psimod	Protein Modification Ontology (Montecchi-Palazzi et al., 2008)
so	Sequence Ontology (Eilbeck et al., 2005)
sbo	Systems Biology Ontology (Courtot et al., 2011)
chebi	Chemical Entities of Biological Interest Ontology (Degtyarenko et al., 2008)
uniprot	UniProt Protein Database (Magrane et al., 2011)
pr	Protein Ontology (Natale et al., 2011)
ro	OBO Relation Ontology (B. Smith et al., 2005)
owl	Web Ontology Language (http://www.w3.org/TR/owl-features)
sbol	The Synthetic Biology Open Language (Galdzicki; Wilson, et al., 2012; Galdzicki; Clancy; Oberortner; Pocock; J. Quinn, et al., 2014)
foaf	Friend of a Friend Vocabulary (http://xmlns.com/foaf/spec)
ipr	InterPro (Mulder et al., 2008)
biopax	Biological Pathway Exchange Ontology Ontology (Demir et al., 2010)

A.2 Terms for Representing Models

Term	Description
Kappa, BioNetGen	Model types.
Agent	Type for declarations of biological entities.

Site	Type for sites of Agents.
State	Type for internal states of Sites.
hasSite, hasState, siteOf, stateOf	Predicates for linking Agents, Sites and States.
Rule	Type for interactions between agents.
hasSubrule, subruleOf	Specifies that a rule has a subrule (i.e., KaSim subrules).
Observable	Type for agent patterns counted by a simulation.

A.3 Terms for Representing Rules

Term	Description
Pattern	Type of a pattern as it appears in a Rule or Observable.
lhs, rhs	Predicates for linking a Rule to its left and right hand side Patterns.
pattern	Predicate for linking an Observable to the patterns that it matches.
agent	Predicate for linking a Pattern and a site within it to the corresponding Agent.
status	Specifies a status of a particular Site (and State) in a Pattern.
isStatusOf, internalState	Predicates for linking a status in a Pattern to corresponding Site and State declarations.
isBoundBy	Specifies the bond that a Site is bound to in a particular Pattern. Bonds are identified via URIs.
BoundState, UnboundState	Terms denoting that a Site in a Pattern is bound or unbound.

A.4 Annotation Predicates

Annotating entities in rule-based models. Terms marked with † are used for machine-generated representations of rules and patterns, and are not usually for annotating models.

Term	Annotation Values
Agent declarations:	
rdf:type	Agent
dct:isPartOf	Identifier for the Model.
hasSite	Identifier of a Site.
biopax:physicalEntity	A biopax:PhysicalEntity term, e.g. DnaRegion or SmallMolecule.
bqbiol:is	A term representing an individual type of an Agent entity, e.g. a protein entry from UniProt.
bqbiol:isVersionOf	A term representing the class type of an Agent entity, e.g. a SO term for a DNA-based agent.
Site declarations:	
rdf:type	Site
hasState	Identifier for an internal state.

bqbiol:isVersionOf	A term representing the type of the site, e.g. A SO term for a nucleic acid-based site or an InterPro term for an amino acid-based site.
Internal state declarations:	
rdf:type	State
bqbiol:is	A term representing the state assignment, e.g. a term from the PSIMOD or the PO.
Rules:	
rdf:type	Rule
dct:isPartOf	Identifier for the Model.
bqbiol:is	A term representing an individual type of a rule, e.g. a KEGG entry.
bqbiol:isVersionOf	A term representing a class type of a rule, e.g. an EC number, a SO term or a GO term.
subrule	Identifier for a Rule entity.
lhs† rhs†	References to the patterns forming the left and right hand side of the rule.
Observables:	
rdf:type	Observable
dct:isPartOf	Identifier for the Model.
pattern†	References the constituent patterns.
Patterns:	
rdf:type	Pattern
ro:hasFunction	A GO term specifying a biological function.
agent†	Reference to the corresponding Agent declaration
internalState†	Reference to a representation of a site's state
isStatusOf†	Reference from a site's state to the corresponding site
Variables:	
rdf:type	sbo:SBO:0000002 (*quantitative systems description parameter*)
dct:isPartOf	Identifier for the Model.
bqbiol:isVersionOf	A term representing a variable type. If exists, the term should a subterm of SBO:0000002.

Appendix B

The Genetic Circuit Compiler Ontology

B.1 GCC Vocabulary Terms

```
# -*- n3 -*-
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

gcc:part a gcc:Token; skos:prefLabel "name".
gcc:Part a owl:Class;
    gcc:tokens gcc:part.

gcc:next a gcc:Token; skos:prefLabel "next".

gcc:transcriptionFactor a gcc:Token;
    skos:prefLabel "transcriptionFactor";
    gcc:default 1.0.
gcc:transcriptionFactorBindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorBindingRate";
    gcc:default 1.0.
gcc:transcriptionFactorUnbindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorUnbindingRate";
    gcc:default 1.0.

gcc:rnapBindingRate a gcc:Token;
    skos:prefLabel "rnapBindingRate";
    gcc:default 1.0.
gcc:rnapDNAUnbindingRate a gcc:Token;
    skos:prefLabel "rnapDNAUnbindingRate";
```

```
    gcc:default 1.0.
gcc:rnapRNAUnbindingRate a gcc:Token;
    skos:prefLabel "rnapRNAUnbindingRate";
    gcc:default 1.0.

gcc:ribosomeBindingRate a gcc:Token;
    skos:prefLabel "ribosomeBindingRate";
    gcc:default 1.0.
gcc:ribosomeRNAUnbindingRate a gcc:Token;
    skos:prefLabel "ribosomeRNAUnbindingRate";
    gcc:default 1.0.
gcc:ribosomeProteinUnbindingRate a gcc:Token;
    skos:prefLabel "ribosomeProteinUnbindingRate";
    gcc:default 1.0.

gcc:transcriptionInitiationRate a gcc:Token;
    skos:prefLabel "transcriptionInitiationRate";
    gcc:default 1.0.
gcc:transcriptionElongationRate a gcc:Token;
    skos:prefLabel "transcriptionElongationRate";
    gcc:default 1.0.

gcc:translationElongationRate a gcc:Token;
    skos:prefLabel "translationElongationRate";
    gcc:default 1.0.

gcc:rnaDegradationRate a gcc:Token;
    skos:prefLabel "rnaDegradationRate";
    gcc:default 1.0.
gcc:proteinDegradationRate a gcc:Token;
    skos:prefLabel "proteinDegradationRate";
    gcc:default 1.0.

gcc:overlaps
    rdfs:domain gcc:Part;
    rdfs:range gcc:Part.

gcc:Operator rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:operator.ka;
    gcc:bnglTemplate rbmt:operator.bngl;
    gcc:tokens
        gcc:transcriptionFactor,
        gcc:transcriptionFactorBindingRate,
        gcc:transcriptionFactorUnbindingRate,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionInitiationRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
```

```
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:Promoter rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:promoter.ka;
    gcc:bnglTemplate rbmt:promoter.bngl;
    gcc:tokens
        gcc:next,
        gcc:rnapBindingRate,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionInitiationRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:RibosomeBindingSite rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:rbs.ka;
    gcc:bnglTemplate rbmt:rbs.bngl;
    gcc:tokens
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeBindingRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:protein a gcc:Token;
    skos:prefLabel "protein".

gcc:CodingSequence rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:cds.ka;
    gcc:bnglTemplate rbmt:cds.bngl;
    gcc:tokens
        gcc:protein,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
```

```
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:Terminator rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:generic.ka;
    gcc:bnglTemplate rbmt:generic.bngl;
    gcc:tokens
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.
```

data/compiler/composition.ttl

B.2 Additional Inference Rules for GCC

```
# -*- n3 -*-
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

## The preferred label of a part is it's part slug
{ ?part gcc:part ?label } => { ?part skos:prefLabel ?label }.

## Derivation of templates
{ ?part a [ gcc:kappaTemplate ?template ] } => { ?part gcc:kappaTemplate ?
    template }.
{ ?part a [ gcc:bnglTemplate ?template ] } => { ?part gcc:bnglTemplate ?
    template }.

## Translation of special predicates to replacement instructions
{ ?kind gcc:tokens ?token .
  ?token skos:prefLabel ?label .
  ?part a ?kind; ?token ?value } =>
{ ?part gcc:replace [ gcc:string ?label; gcc:value ?value ] }.

## overlaps is a symmetric relation
{ ?a gcc:overlaps ?b } => { ?b gcc:overlaps ?a }.
```

data/compiler/composition.n3

B.3 Complete Model of the Elowitz Repressilator

```
# -*- n3 -*-
@prefix : <http://id.inf.ed.ac.uk/rbm/examples/repressilator#>.
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

## Top-level model description.
:m a rbmo:Model;
    ## bibliographic metadata
    dct:title "The Elowitz repressilator constructed from BioBrick parts";
    dct:description "Transcription of the treatment of the Elowitz
    repressilator given in the Kappa BioBricks Framework book chapter";
    rdfs:seeAlso <http://link.springer.com/protocol/10.1007/978
    -1-4939-1878-2_6>;
    gcc:prefix <http://id.inf.ed.ac.uk/rbm/examples/repressilator#>;
    ## include the host environment
    gcc:include <host.ka>;
    ## The expression of the model as a genetic circuit
    gcc:circular (
        :R0040o :R0040p :B0034a :C0051 :B0011a
        :R0051o :R0051p :B0034b :C0012 :B0011b
        :R0010o :R0010p :B0034c :C0040 :B0011c
    ).

:P0040 a gcc:Protein;
    skos:prefLabel "P0040";
    rdfs:label "TetR".

:P0051 a gcc:Protein;
    skos:prefLabel "P0051";
    rdfs:label "lambda-Cl".

:P0010 a gcc:Protein;
    skos:prefLabel "P0010";
    rdfs:label "LacI".

:C0051 a gcc:CodingSequence;
    rdfs:label "Coding sequence for lambda-Cl";
    gcc:part "C0051";
    gcc:protein :P0051;
    gcc:proteinDegradationRate 0.0001.
```

```
45
46 :C0012 a gcc:CodingSequence;
47     gcc:label "Coding sequence for LacI";
48     gcc:part "C0012";
49     gcc:protein :P0010;
50     gcc:proteinDegradationRate 0.0001.
51
52 :C0040 a gcc:CodingSequence;
53     gcc:label "Coding sequence for TetR";
54     gcc:part "C0040";
55     gcc:protein :P0040;
56     gcc:proteinDegradationRate 0.0001.
57
58 :B0034a a gcc:RibosomeBindingSite;
59     rdfs:label "Ribosome binding site";
60     gcc:part "B0034a".
61
62 :B0011a a gcc:Terminator;
63     rdfs:label "Terminator, stop codon";
64     gcc:part "B0011a".
65
66 :B0034b a gcc:RibosomeBindingSite;
67     rdfs:label "Ribosome binding site";
68     gcc:part "B0034b".
69
70 :B0011b a gcc:Terminator;
71     rdfs:label "Terminator, stop codon";
72     gcc:part "B0011b".
73
74 :B0034c a gcc:RibosomeBindingSite;
75     rdfs:label "Ribosome binding site";
76     gcc:part "B0034c".
77
78 :B0011c a gcc:Terminator;
79     rdfs:label "Terminator, stop codon";
80     gcc:part "B0011c".
81
82 :R0040o a gcc:Operator;
83     rdfs:label "TetR activated operator";
84     gcc:part "R0040o";
85     gcc:transcriptionFactor :P0040;
86     gcc:transcriptionFactorBindingRate 0.01;
87     gcc:transcriptionFactorUnbindingRate 0.01.
88
89 :R0040p a gcc:Promoter;
90     rdfs:label "TetR repressible promoter";
91     gcc:part "R0040p";
92     gcc:next "B0034a";
93     gcc:rnapBindingRate [
```

```
      gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0040o] );
      gcc:value 7e-7
    ], [
      gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0040o] );
      gcc:value 0.0007
    ].

:R0051o a gcc:Operator;
    rdfs:label "lambda-Cl activated operator";
    gcc:part "R0051o";
    gcc:transcriptionFactor :P0051;
    gcc:transcriptionFactorUnbindingRate 0.01;
    gcc:transcriptionFactorBindingRate 0.01.

:R0051p a gcc:Promoter;
    rdfs:label "lambda-Cl repressible promoter";
    gcc:part "R0051p";
    gcc:next "B0034b";
    gcc:rnapBindingRate [
        gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0051o] );
        gcc:value 7e-7
    ], [
        gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0051o] );
        gcc:value 0.0007
    ].

:R0010o a gcc:Operator;
    rdfs:label "LacI activated operator";
    gcc:part "R0010o";
    gcc:transcriptionFactor :P0010;
    gcc:transcriptionFactorBindingRate 0.01;
    gcc:transcriptionFactorUnbindingRate 0.01.

:R0010p a gcc:Promoter;
    rdfs:label "LacI repressible promoter";
    gcc:part "R0010p";
    gcc:next "B0034c";
    gcc:rnapBindingRate [
        gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0010o] );
        gcc:value 7e-7
    ], [
        gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0010o] );
        gcc:value 0.0007
    ].
```

data/compiler/repressilator.ttl